Transevol: A Tool to Evolve Legacy Model Transformations by Example

Joseba A. Agirre, Goiuria Sagardui and Leire Etxeberria
Department of Computing, Mondragon University, Loramendi, Mondragon, Spain

Keywords: Model Driven Development, Model Transformation Development, Model Transformation by Example, Model Transformation Execution Trace, Model Differences.

Abstract: The use of Model Driven Development (MDD) approach is increasing in industry. MDD approach raises the level of abstraction using models as main artefacts of software engineering processes. The development of model transformations is a critical step in MDD. Tasks for defining, specifying and maintaining model transformation rules can be complex in MDD. Model Transformation By Example (MTBE) approaches have been proposed to ease the development process of transformation rules. MTBE uses pair of input/output models to define the model transformation. Starting from pairs of example models the transformation rules are derived semi-automatically. The aim of our approach is to derive the adaptation operations that must be implemented in a legacy model transformation to fulfil a new transformation requirement. An MTBE approach and a tool to develop and evolve ATL transformation rules have been developed. Our approach derives the transformations operations automatically using execution traceability data and models differences. The developed MTBE approach can be applied to evolve legacy model transformations. A real case study is introduced to demonstrate the usefulness of the tool.

1 INTRODUCTION

Model transformations are fundamental in Model Driven Development (MDD). A model transformation takes input models conforming to the source metamodel and produces output models conforming to the target meta-model. To express meta-models and models several tool exist, for example the popular Eclipse Modeling Framework (EMF). On MDD, a model transformation is specified through a set of transformation rules, usually using transformation languages such as Atlas Transformation Language (ATL) (Jouault et al., 2008), QVT (OMG, 2011) or EPSILON (Kolovos et al., 2008). There are two kinds of model transformations: endogenous and exogenous (Mens, and Van Gorp, 2006). *Endogenous transformations* are transformations between models expressed with the same meta-model. *Exogenous transformations* are transformations between models expressed using different meta-models. Tasks for defining, specifying and maintaining transformation rules are usually complex and critical in MDD.

In order to facilitate the development of transformations rules reuse mechanisms (Wimmer et al., 2012), reusable transformation design patterns (Iacob et al., 2008) and refactoring operations (Wimmer et al., 2012) have been described. Model Transformation By Example (MTBE) (Varró, 2006) approaches have been proposed to ease the development process of transformation rules. By-example approaches define transformations using examples models. In MTBE starting from pairs of example input/output models the transformation rules are derived. Different MTBE approaches exists (Kappel et al., 2012). MTBE approaches for model transformation are classified in two types (I) demonstration based and (II) correspondences based. *Model transformation by demonstration* (MTBD) (Sun et al., 2009) specifies the desired transformation using modifications performed on example models. *MTBE based on correspondences*, uses pairs of input/output models and a mapping data between them to derive the transformation rules. MTBE approaches allow to specify the model transformation using models, which is very intuitive (Varró, 2006). Transformation rules are generated semi-automatically so the model transformation development process is improved (Sun et al., 2009).

Agirre J., Sagardui G. and Etxeberria L..
Transevol - A Tool to Evolve Legacy Model Transformations by Example.
DOI: 10.5220/0004999702340245
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 234-245
ISBN: 978-989-758-036-9
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

Examples models can also be used to test the implemeted transformations (Kappel et al., 2012).

Evolving legacy model transformations is a complex task. The aim of the approach is to reduce maintenance efforts when modifications are required in a model transformation. Our approach is focused on generating semi-automatically transformation rules from pairs of example input/output models and transformation rules execution traces (see figure 1). The main characteristic of the approach is that it can be used to evolve legacy model transformations. Models differences, obtained after modifying a source model and a target model, are the core of the approach. Concretely the expected output model and the present produced output model, when transformation is executed, are compared. And adaptations of the transformation implementation are derived.

In this paper, we present a JAVA & EMF tool (TransEvol) for semi-automatic derivation of ATL model to model transformations to fulfill a new transformation requirement. This paper provides the following contributions to the study of MTBE (I) MTBE approach based on model differences for exogenous and endogenous model transformations, (II) a MTBE approach applicable to ATL legacy model transformations and (III) Validation of the usefulness of the approach in a real legacy model transformation using a tool that we have developed.

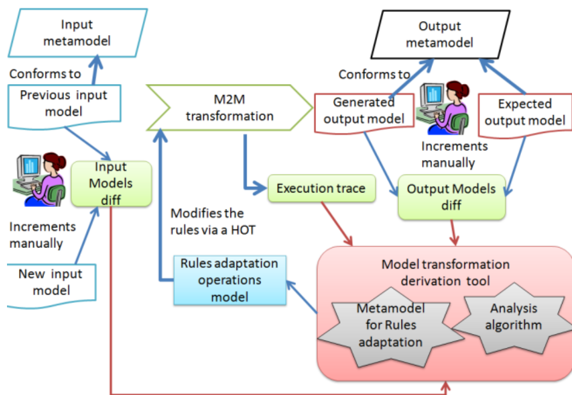

Figure 1: Approach for automatic model transformation analysis to derive adaptation operations.

In the following sections we detail the solution which guides the implementation of model transformations. First, in section 2, the legacy model transformation example that motivated the need for automating the evolution of trasformation rules is presented. Section 3 describes the MTBE approach used for the model transformation development. The fourth resumes the results of applying the approach on the motivating example. Then in section 5 a brief description of the related work is presented. Finally in section 6 the conclusions and future work are resumed.

2 LEGACY MODEL TRANSFORMATION EXAMPLE

In (Agirre et al., 2012) a MDD code generation system is presented. The MDD system generates ANSI-C code from component-based SW architectures, previously designed in UML. The MDD system generates the C code in two steps. As in Model Driven Architecture (MDA) (Mellor, 2004) platform independent models (PIM) are transformed into platform specific models (PSM), and finally the PSM is transformed in code. In our case study, UML designs are transformed to intermediate models representing ANSI-C code through a model to model (M2M) transformation. SIMPLEC (Agirre et al., 2010) meta-model is used to represent a subset of ANSI-C. The exogenous M2M transformation is implemented using ATL transformation language. Once the SIMPLEC models are obtained, a model to text (M2T) transformation is applied to SIMPLEC models to generate ANSI-C code. XPAND2 (Open Architecture Ware, 2010) based templates are used to generate the output source code. Figure 2 resumes the example MDD code generation system.

The M2M transformation is composed by 8 ATL modules with 73 transformation rules and 44 helper functions. The M2T transformation has 31 templates to generate the ANSI-C code from SIMPLE-C models. Originally, the MDD system of the case study did not offer concurrency characteristics at the design model and at the generated code. At one point, to add concurrency capabilities was required. This kind of situation is defined as abstraction evolution (Van Deursen et al., 2007). In abstraction evolution new domain concepts must be added to the MDD system, so several artefacts of the MDD system are affected. In this case, the source metamodel (UML) does not support the abstractions required to offer concurrency, so it is necessary to extend the metamodel or to add a new metamodel.

The UML MARTE (Modeling and Analysis of Real-Time and Embedded Systems) (OMG, 2009) profile was selected to add concurrency concepts in the design models. MARTE profile is an UML extension that provides support for specification, design, and verification of real time and embedded

systems in UML. Due to the division of the generation in two stages the M2T transformation and the SIMPLEC metamodel did not require any change. Obviously, the source metamodel extension implies a co-evolution of the M2M transformation. The only documentation available about the M2M transformation was a few input models, so an exhaustive navigation was required to adapt manually the complex M2M transformation.

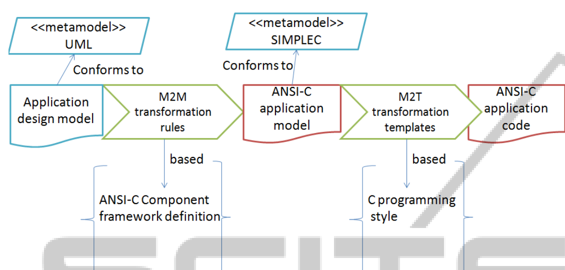

Figure 2: UML to C MDD code generation system.

Our approach is based on defining the desired transformation by editing a previous input model and demonstrating the changes in transformations that lead to a target model. To relate the differences to a legacy model transformation an execution trace data is required. Combining the example models data with the transformation execution trace data the adaptation operations that must be implemented in the legacy model transformations are derived automatically (see figure 1). This way the development time is reduced and the probability to incur in errors is reduced. A JAVA & EMF tool has been developed to deduce automatically the adaptation operations on ATL transformation rules. The tool implements an algorithm to derive adaptation operations using model differences and execution trace data. A metamodel to express adaptation operations on transformation rules have been defined.

3 OUTPUT MODELS DIFFERENCES DRIVEN MODEL TRANSFORMATION ANALYSIS

The aim of the approach is to derive the adaptation operations that must be implemented in a legacy model transformation to fulfil a new requirement. Starting from pairs of example input/output models the tool deduces a number of adaptations in the transformation. The transformation analysis process consists of the following phases (see figure 1):

1. Adapt manually a previous input model to add the new requirement and obtain the differential model between both models (For example, the addition of a UML MARTE task model to express the concurrency).
2. Adapt manually a previous output model to add the requirement and obtain the differential model between the both models (For example, adding SIMPLEC elements that represents the implementation of the designed task model).
3. Obtain the traceability between the previous design model, the generated output model and the transformation rules.
4. Deduce the adaptations to be made in the transformation rules to fulfil the new transformation requirement using TransEvol.
5. Execute a Higher Order Transformation (HOT) to semi-automatically adapt the transformation rules.
6. Manually finish the transformation rules implementation.
7. Validate the transformation implementation using the manually generated input and output model.

Figure 3, 4 and 5 show an example of the artefacts that take part in the analysis process. Some details of the case study have been omitted in the interest of improving the understability. First, the input model is modified manually to add a task model with three periodic tasks to the example design (see figure 3). To specify the model transformation the ouput model must be modified to integrate the SIMPLEC elements that correspond to the designed task model. Three new methods are added to the ouput model (see figure 4). Using this information, the approach detects an one-to-one mapping and a new matched rule must be generated. To bind the new output elements with its container element a binding statement also must be created in the rule that generates the container. Figure 5 lists the resulting transformation rules.

The transformation rules analysis tool, TransEvol, relates EMFDiff (Toulmé, 2006) differences types of the output models to adaptation operations to apply on the model transformation. The tool implements an algorithm that derive adaptation operations from the difference model

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

between a model generated by the M2M transformation (GOm, Generated output model) and an expected output model (EOm)).

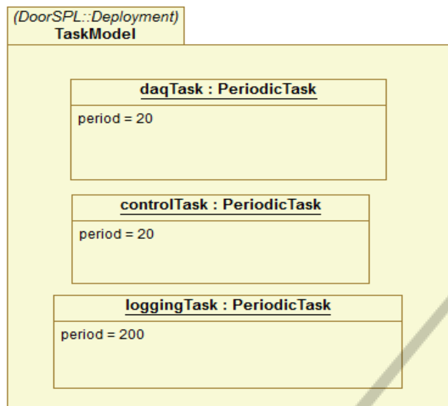

Figure 3: The task model aggregated to the example design model.

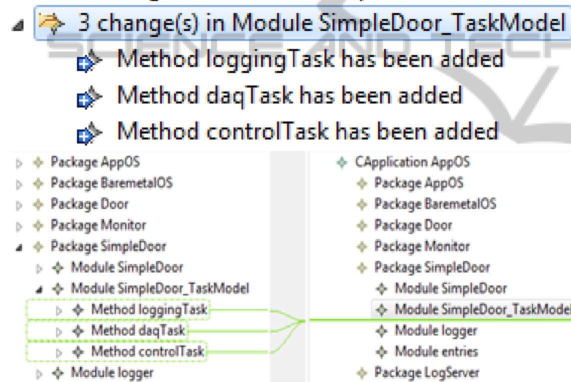

Figure 4: The output model differences due to the task model.

```
rule PeriodicTask2Method {
    from
        i: Uml!InstanceSpecification
    to
        m : SimpleC!Method(
            name <- i.name
            ,code <- thisModule.periodicTaskCode(i)
    )
```

new matched rule

```
rule ComponentTaskModelInterface{
    from
        r: Uml!Realization ( r.supplierIsTaskModel() )
    to
        m : SimpleC!Module(
            name <- r.client->first().name+'_TaskModel'
            ,path <- r.client->first().name
            ,methods <-  r.getPeriodicTasks()
            ,externals <- r.client->first().getKomponentModule()
        )
}
```

new binding statememt in a legacy transformation rule

Figure 5: The required adaptation operations for the model transformation to integrate the task model concepts.

This difference model is called Output models differential (**ΔOm** = EOm – GOm) and is generated using EMFCompare (Brun and Pierantonio, 2008) and conforms to EMFDiff metamodel. The EMFDiff metamodel types used to analyze the model transformation are: addition of an element (*ModelElementChangeLeft*), removal of an element (*ModelElementChangeRight*), change of an element container (*MoveModelElement*), addition of an attribute (*AttributeChangeLeftTarget*), addition of a reference (ReferenceChangeLeftTarget), modification of a reference (*UpdateReference*) and modification of an attribute (*UpdateAttribute*). The EMFDiff differences offer basically the data of the new element, the deleted or updated element, the element affected by the change and the container of the new element.

3.1 Specifying Adaptation Operation for the Transformation Rules

TransEvol tool uses a metamodel called MMRuleAdaptation (figure 3) to express the required adaptation operations for the transformation rules. The transformation rules are subject to the following refinement modifications: *addRule*, *splitRule*, *deleteRule*, *deleteOutputPatternElement*, *deleteBinding*, *addInputPatternElement* *addOutputPatternElement*, *addBinding*, *moveOutputPatternElement*, *moveBinding*, *updateBinding*, *UpdateFilter* and *UpdateSource*. After the analysis, the tool generates a model expressing the required adaptation. Any modification operation is defined as an AdaptationTarget. Each adaptation target has a set of adaptation operations. Each adaptation operation requires different information to specify the modification, see table 1. The metamodel uses ATL metamodel elements to express the data related to each modification operation. Table 1 collects the data required to express each adaptation operation.

3.2 Relationship between Emfdiff Difference Types and Adaptation Operations

The tool relates EMFDiff differences types of the output models with adaptation operations. Table 2 resumes the relation between EMFDiff types and adaptation operations. Not always the same difference type instance is related to the same adaptation operation.

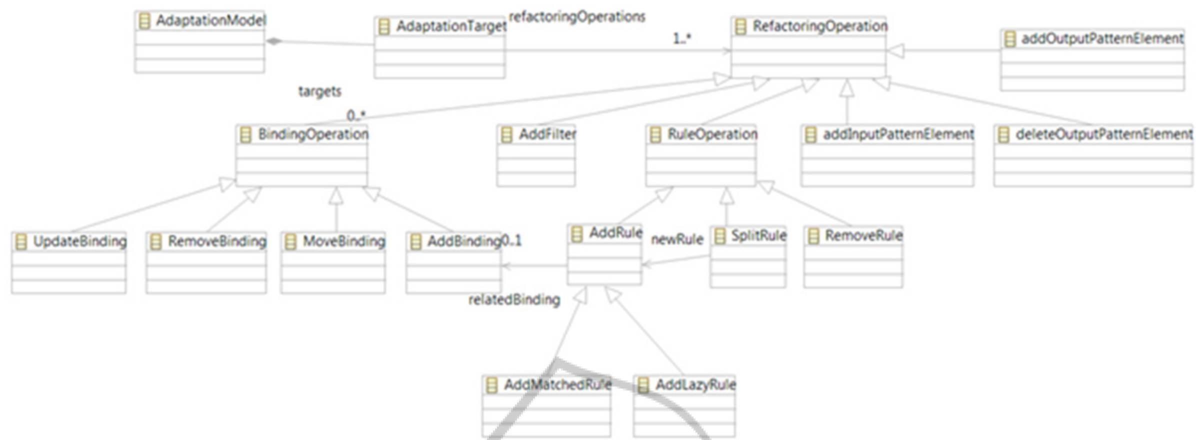

Figure 6: MMRuleAdaptation metamodel.

Table 1: MMAdapatationRule metamodel's adaptation operations.

Adaptation operation	Required Data
Add Rule	newRule*: ATL!Rule* relatedBinding: *MMRuleAdaptation!AddBinding*
Add MatchedRule (extends addRule)	newRule*: ATL!Rule* relatedBinding: *MMRuleAdaptation!AddBinding*
Add LazyRule (extends addRule)	newRule*: ATL!Rule* relatedBinding : *MMRuleAdaptation!AddBinding*
Split Rule	affectedRule: *ATL!Rule* newRule: *MMRuleAdaptation!AddRule*
Add Binding (extends BindingOperation)	affectedRule: *ATL!Rule* newBinding : *ATL!Binding*
Remove Binding (extends BindingOperation)	affectedRule: *ATL!Rule* affectedBinding: *ATL!Binding*
Update Binding (extends BindingOperation)	affectedRule: *ATL!Rule* affectedBinding: *ATL!Binding* newValue:OCL!OclExpression
Move Binding (extends BindingOperation)	affectedRule: *ATL!Rule* toRule: *ATL!Rule* binding: *ATL!Binding*
Add filter to input pattern	newFilter: *OCL!OclExpression* affectedRule: *ATL!Rule*
Add input pattern element	affectedRule: *ATL!Rule* *newInput:ATL!InputPatternElement*
Add output pattern elemen	affectedRule: *ATL!Rule* *outputPattern: ATL!OutputPatternElement*
Delete out pattern element	affectedRule: *ATL!Rule* *outputPattern: ATL!OutputPatternElement*

3.3 Model Transformation Analysis Algorithm

Using only the output models differences the adaptations operations cannot be deduced correctly, (for example the affected rule in a binding could not be obtained). More information is required to deduce the operations. The execution traceability data (**ETr**) is fundamental for the automatic deduction of the transformation rules modifications. ATL2Trace (Joault, 2005) Higher-Order Transformation (HOT) is applied to the transformation under development to obtain the execution trace. Using the traceability information the algorithm can deduce not only the new transformation rules, also the adaptation operations to apply on the legacy transformation rules. To derive the modification operations also the input models differential (**ΔIm**) must be used. Additionally input and output metamodel class coverage (Fleurey et al., 2009) is required. The metamodel class coverage represents each meta-class is instantiated at least once. The algorithm uses the differential of the input metamodel class coverage (**ΔImc**) due to ΔIm. And also uses the output metamodel class coverage differential (**ΔOmc**) due to **ΔOm**. Combining the ΔOm, ΔIm, ETr, ΔImc and ΔOmc the algorithm obtains the modifications that must be done to adapt the transformation rules for the new transformation requirement.

To specify a model transformation example ΔOm, ΔIm, ETr, ΔImc and ΔOmc models are required. Each difference element is related to an adaptation operation depending on the difference type. The analysis algorithm fits correctly to scenarios that have **ΔImc= 0** or **ΔImc**=1 and **ΔOmc>0**. When **ΔImc** is higher than one we recommend to divide the examples in a set of **ΔImc=1** examples.

The algorithm first takes a difference element of the ΔOm and decides which kind of difference is:

1. Addition of output model elements
2. Removal of an output model element
3. Change of an element container
4. Addition and modification of attributes
5. Addition and modification of references

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 2: Relationship between EMFDiff metamodel types and adaptation operations for model transformations.

EMFDiff difference type	EMFDiff type description	Adaptation operations
ModelElement ChangeLeft	Addition of an element	Add matched rule and add binding
		Add lazy rule and add binding
ModelElement ChangeRight	Removal of an element	Add filter
		Remove rule
MoveModelEleme nt	Change of container	Split rule and modify binding
		Move binding
ReferenceChange LeftTarget	Addition of a reference	Add binding
UpdateReference	Update of a reference value	Update binding
		Add input pattern
AttributeChange LeftTarget	Addition of an attribute value	AddBinding
UpdateAttribute	Modification of an attribute value	Add binding
		Update binding
		Add input pattern

Once the type of the difference is decided, the algorithm must deduce the modification that must be applied to the model transformation. Depending on the scenario of the model transformation the adaptation operation for an output EMFDiff difference type may be slightly different. For example when some elements are added to the output model (**ΔOm>0**) a matched rule, a lazy rule or an output pattern element must be added, and also a binding must be created to associate the new element with a previously created model element. The addition of an output model element can be due to a one-to-one mapping, one-to-many mapping (different output elements types), one-to-many mapping (same output elements type) or many-to-many mapping. Depending on the scenario of the element addition the adaptation operations vary. To select the scenario the tool uses the ΔOm, ΔIm, ΔImc and ΔOmc models data.

3.3.1 Addition of Elements: One-to-One Mapping Scenario

The conditions to detect a one-to-one mapping scenario are: (I) The number of ModelElementChangeLeft in the ΔIm and the ΔOm is equal to 1, (II) the metamodel class coverage increment for the input and output metamodel must be 1. This scenario requires a new matched rule. The adaptation operation of adding a new matched rule is compound by a new rule and a binding. The data required to define the new matched rule is the Input Pattern element, the output pattern element and the rule name. The input pattern element is the type of any of the added element of the ΔIm model. The output pattern element is the type of one of the added element of the ΔOm. The rule name is the concatenation of both types. To create the binding that relates the new target element to its container the rule that created the container element must be searched. To search the rule that creates the container the execution traceability data is used. Once the affected rule is founded the binding statement is established.

3.3.2 Addition of Elements: One-to-Many Mapping Scenario

The second kind of scenario is related to one-to-many mappings. This kind of scenarios requires the creation of a new output pattern element or a new lazy rule. If different types of target elements are created new output pattern elements are added to a rule. If instances of the same type are created for an input element type lazy rules are required. The transformation examples can be specified differently, table 3 resumes one-to-many scenarios that the algorithm detects.

3.3.3 Addition of Elements: Many-to-Many Mapping Scenario

Many-to-many scenario is defined when a set of elements are added and both ΔImc and ΔOmc are higher than one. Two strategies can apply to this scenario. The first strategy is to specify the transformation example with a set of one-to-many mapping examples, where ΔImc is equal to 1 in each step. When ΔImc is greater than 1 the algorithm must align input elements with output elements using the similarity of its properties values. In those cases false positives adaptation operations can be deduced. For those cases a warning message is used. That way the transformation rules developers can analyse the adaptation operation model proposal and change it manually.

3.3.4 Removal of an Output Model Element

Two removal scenarios are detected by the algorithm. A matched rule is removed when ΔOmc = -1. The other scenario occurs when ΔOmc = 0 and some *ModelElementChangeRight* appears (see table 4). This scenario requires a filtering operation in the input pattern element. In both cases the affected rule is founded searching in the execution trace the rule that generates the removed elements.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 3: One-to-many mapping scenarios.

Previous transformation	Desired transformation	Scenario data	Adaptation operation
		$\Delta Im = 0$ $\Delta Om = N$ $\Delta Imc=0$ $\Delta Omc = 1$	Add output pattern element
		$\Delta Im < \Delta Om$ $\Delta Imc=1$ $\Delta Omc = N$	Add matched rule with multiple output pattern elements
		$\Delta Im < \Delta Om$ $\Delta Imc=1$ $\Delta Omc = N$	Add matched Rule with multiple output pattern elements
		$\Delta Om>0$ $\Delta Imc=0$ $\Delta Omc = N$	Add output pattern element
		$\Delta Im < \Delta Om$ $\Delta Imc=1$ $\Delta Omc = 1$	Add matched rule Add Lazy rule
		$\Delta Im < \Delta Om$ $\Delta Imc=0$ $\Delta Omc = 0$	Add Lazy rule

Legend:
- Arrow: Transformation
- Geometric shapes (left side of the arrow): Elements of the input model
- Geometric shapes (right side of the arrow): Elements of the Output model

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 4: Removing output elements.

Previous transformation	Expected transformation

Legend:
- Arrow: Transformation
- Geometric shapes (left side of the arrow): Elements of the input model
- Geometric shapes (right side of the arrow): Elements of the Output model

3.3.5 Change of an Element Container

Sometimes without any modification in the input models ($\Delta Imc = 1$ and $\Delta Im=0$) the model transformation evolves and requires to change the instance of the container of an output element or even the container type. Both scenarios are detected by the algorithm. The first scenario involves a split rule operation. To split the affected rule a copy of the rule is done but filtering is added to the input pattern and a binding must be modified. When the type of the container changes a binding must be deleted in the rule that created the previous container and a binding must be added in the rule that created the desired container. To search those affected rules the execution trace of the previously executed transformation is used.

3.3.6 Addition and Modification of Attributes or References

The operations related to these scenarios are modification of a binding or an addition of a binding. In these cases, the execution trace is used to search the affected rule. The information of the output elements that have the difference (*Updateattribute*, *UpdateReference*, *ReferenceChangeLeftElement* and *AttributeChangeLeftElement*) is used to search the

affected rule in the traceability data and to define the binding statement.

3.3.7 the Algorithm: Summary

Using the differences models and the traceability information the analysis of the transformation can be done. The difference model is based on model elements and not on metamodel elements, so several differences may be referred to the same change to be made in the transformation rules. We therefore must filter the adaptation operations to obtain the final adaptation operation model. Figure 4 represents a simplification of the algorithm.

4 IMPLEMENTING THE ADAPTATION OPERATIONS

Once the adaptation operations model is generated the last step is to implement and validate the adaptation operations applied to the transformation rules. A HOT has been implemented to perform automatically the adaptation operations on the ATL module. The HOT takes as input the ATL module and the adaptation model. Despite the tool can detect the listed operations actually the HOT only implements addRule, addBinding , splitRrule and addFilter. The operations that are not executed by the HOT must be implemented manually. Once the transformation rules are adapted the new input model and the desired output models are used to validate the implementation of the transformation rule.

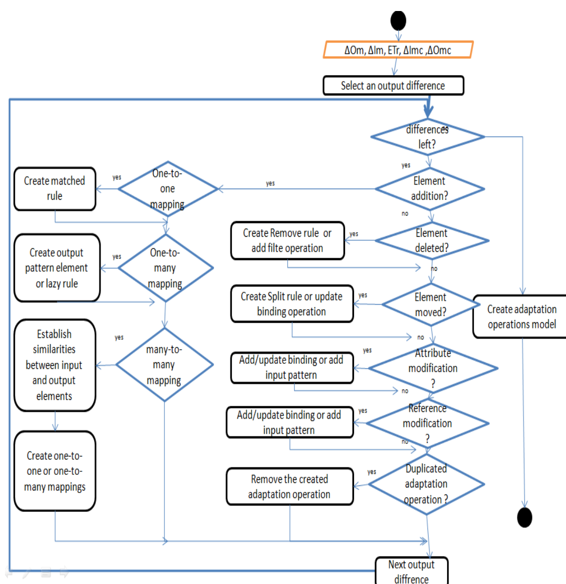

Figure 6: Algorithm for adaptation operations deduction

5 VALIDATION OF THE APPROACH

During the development of the tool several small model examples were used to validate the detection and generation of the different adaptation operations. Those examples are toy examples so a legacy model transformation was required to test the approach in a real context. For a first validation of the tool in a real context the case study presented in section 2, a model transformation from UML to SIMPLEC, was chosen. Following the result of applying the tool to the case study will be shown. Then the threats to the validity are listed.

5.1 Applying the Tool to the Case Study

In this subsection, results from applying the tool to the M2M transformation that generates SIMPLEC models, representing C source code, from UML SW designs, is presented. The M2M transformation is implemented in ATL. The M2M transformation is performed incrementally by superimposition mechanism of ATL (Wagelaar et al., 2009). The new requirement was to add concurrency capabilities to the generated code. As presented before, to achieve this objective, UML MARTE profile was selected and the complex M2M transformation (8 files, 40 matched rules, 30 lazy rules and 44 helper functions) required some changes.

To apply the tool a previously used UML design was selected: a UML design of an automatic door controller without concurrency. The M2M transformation was executed to generate the output model. Also the transformation execution trace model was generated. To start with the analysis, using UML MARTE a task model was added to the automatic door controller design. The API selected to express concurrency was a bare-metal API similar to FreeRTOS API. On the next step, the expected target output model with concurrency was created changing manually the generated output model. Finally, the difference models between the original and the incremented models were generated using EMFCompare Tool. A total of 13 differences were detected between the input models and 12 differences were detected on the output models.

Instead of specifying all the differences in one step the transformation example was divided in two steps. (I) the platform provider, the concurrency API, was specified as MARTE describes, (II) the

task model was designed and each task was related to its behaviour.

In the firts step, the concurrency API model (two functions: *addTask* and *schedule*) was defined using MARTE stereotypes in the design model and a header with the API definition was added to the output model. The scenario was: ΔImc=3, ΔOmc=0, ΔIm=4 and ΔOm=3. Three one-to-one mappings were detected, so three matched rules were deduced in this step: (I) the generation of the header of the API model (II) the *addTask* function and (III) the *schedule* function.

The second step requires the creation of the task model. And also the assignment of the behaviour to each task. In this case the scenario was: ΔImc=3, ΔOmc=0, ΔIm=10 and ΔOm=10. Seven of the ΔOm differences were *ReferenceChangeLeftTarget* type. The remaining three differences were addition of output elements. The adaptation operations deduced were four *addBinding* operations, that affected legacy transformation rules, and two *addMatchedRule* due to two one-to-many mappings detected. The models and the result corresponding to the task model can be seen in figure 3, 4, and 5.

When the tool derives add rule operations also a binding statement to attach the new elements with the container is derived. In those cases, the binding statement expression is implemented by a helper function. The tool generates the helper header definition and the call statement. The algorithm of the helper functions is completed manually.

The final model transformation implementation was validated applying the transformation to the new design model and comparing the new generated model with the expected model. All the deduced adaptation operations were correct. To apply the tool it is enough knowing the changes that are necessary in the M2M transformation input and output models. Previous knowledge of the model transformation implementation is not required, so the time required to adapt the M2M transformation is reduced.

5.2 Threats to Validty

The proposed case study is a real system and thus do not consider a certain number of factors that could affect the validation of the method:

- *Correctness*: Although initial case study show promising results, as all the transformation rules have been correctly identified, algorithm should be proved in more complex and different examples to improve the coverage of the validation.
- *Scalability*: The selected case study has legacy transformations (8 files, 40 matches rules, 30 lazy rules and 44 helper functions) and we deployed 13 differences in input models and 12 differences in output models. Although the case study is a real system, validation with bigger case studies is required.
- *Negative constructions*: the algorithm supports the remove matched rule operation and the add filter operation. In this real case study there are not negative constructions. However, the negative constructions have been proved with toy examples during the development of the tool.
- *Many-to-many mapping*: In the case study there are not many to many mappings. At present the tool can detect many to many mappings. However some ambiguities occurs generating the adaptation operations using the tool. To deal with many to many mappings the transformation must be specified with a set of one to many mapping examples.

6 RELATED WORK

The presented approach is highly related to MTBE. There are previous MTBE approaches which already deal with automatic generation of model transformations starting from pairs of example models. Most of the approaches are based on formal mapping to derive the transformations (Bhalog and Varró, 2009). (Strommer and Wimmer, 2008) approach uses correspondence model between input and output model to generate ATL transformation rules. Instead offering a mapping model (García-Magariño et al., 2009) annotates with extra information the source metamodel and the target metamodel to derive the required ATL transformation rules. Our approach also creates ATL transformation rules but a mapping between the desired input and output model or extra information besides the models differentials is not required.

In (Faunes et al., 2013) a genetic programming based approach to derive model transformation rules (implemented with JESS) from input/output models is presented. This approach does not require fine-grained transformation traces. But due to the nature of the search algorithm the approach cannot be used to evolve a legacy model to model transformation. Something similar occurs with (Kessentini et al., 2012) where a heuristic algorithm is used to generate

a new transformed model by similarity with other transformation example models. This approach is a self-tunning transformation so it cannot be used with legacy model transformations.

MTBD are based on defining the desired transformation by editing a source model and demonstrating the changes that evolve to a target model. Most of the MTBD are used on endogenous model transformation (Sun and Gray, 2013) not as MTBE based on correspondences, which can be used with exogenous transformations. (Langer et al., 2010) presents a MTBD approach that can be applied to exogenous model transformation. This approach uses a state-based comparison to determine the executed modification operations after modelling the desired transformation. Using an incremental approach, in each step using a small transformation rule demonstration, internal templates representing the transformation rules are created. Once all the steps are done the templates are transformed to ATL transformation rules. This approach offers an interactive step where the developer can annotate the templates prior to generate the ATL rules to add information about the matching strategies. Because the approach uses templates created by transformation rules demonstrations it is not easy to apply this approach to legacy model transformations. Negative application conditions as well as many-to-one attribute correspondences are not considered. Our approach derives the transformations operations automatically using execution traceability data and models differences. This way the approach can be used to evolve legacy model transformations. (Levy and Muniz, 2013) proposes a similar approach but the generation of the transformation must be done manually.

Most example-based approaches are constructive, that is, the new information always imply adding new elements to the artefact (a transformation in this case). Deleting is more complex. The presented approach can deal with negative constructions.

Metamodel and transformation co-evolution solution also exists. In (Garcia et al., 2012) input metamodel differences are used to derive the evolution on the transformation rules. In (Iovino et al., 2012) weaving between metamodels and transformation rules is used to analyze the impact on the transfortion rules due to input metamodel evolution. These works only derives the modification on the transformation rules when regular metamodel evolution, as attribute modification or metaclass rename, occurs. When new elements on the input metamodel appear, the approach cannot derive the transformation rules.

Most of the MTBE cannot be applied to legacy model transformations. The main contribution of our MTBE approach is that it can be applied to evolve ATL legacy model transformations. Our approach can be applied to both exogenous and endougenous model transformations. We also have validated our approach in a real case study.

7 CONCLUSIONS AND FUTURE WORK

An MTBE approach and a tool to evolve ATL transformation have been presented. A metamodel for expressing adaptation operations for transformation rules and the algorithm to derive the adaptation operations for M2M transformations have been described. The tool has been used successfully for adapting exogenous legacy model transformations to new transformation requirements. The tool generates semi-automatically adaptation operations for ATL transformation rules. The helpers used in the binding statements are only defined and called. The implementation of the helper functions must be done manually. The algorithm used to derive adaptation operation and the metamodel used to express the operations can be used to express operations for transformation languages such as QVT or EPSILON. The tool may require some changes to work with other transformation languages execution traces and also a new HOT, must be implemented for each transformation language.

The algorithm can detect one-to-one, one-to-many and many to many mappings. Negative construction examples are also detected. Actually the derivation of many to many and many to one mappings requires manual intervention. The tool uses output models differentials and execution trace data. In (Matragkas et al., 2013) the same data is used to locate the implementation errors in transformation rules implemented with EPSILON. The tool can be used with that orientation but must be analyzed how.

Once the functionality of the tool has been tested with small examples and a medium real legacy system, more validation on scalabilty and correcteness are required. Currently, we are working on the definition of a methodology for the specification of correct example models. In short-term the tool is going to be used in a legacy model

transformation to aggregate some security requirements to the output models as in (Sun et al., 2013).

ACKNOWLEDGEMENTS

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

Agirre J., Sagardui, G., Etxeberria, L., 2010. Plataforma DSDM para la Generación de Software Basado en Componentes en Entornos Empotrados. *JISBD (pp. 7-15).*

Agirre, J., Sagardui, G., Etxeberria, L., 2012. A flexible model driven software development process for component based embedded control systems. *III Jornadas de Computación Empotradas JCE, SARTECO.*

Balogh, Z., Varró, D., 2009. Model transformation by example using inductive logic programming. *Software and System Modeling 8(3): 347-364*.

Brun, C., Pierantonio, A., 2008. Model differences in the Eclipse modelling framework. EJIP.

Faunes, M., Sahraoui, H., Boukadoum, M., 2013. Genetic-Programming Approach to Learn Model Transformation Rules from Examples. *In Theory and Practice of Model Transformations* (pp. 17-32). Springer Berlin Heidelberg.

García, J., Diaz, O., Azanza, M., 2013. Model transformation co-evolution: A semi-automatic approach. *In Software Language Engineering* (pp. 144-163). Springer Berlin Heidelberg..

García-Magariño, I., Gómez-Sanz, J. J., Fuentes-Fernández, R., 2009. Model transformation by-example: an algorithm for generating many-to-many transformation rules in several model transformation languages. *In Theory and Practice of Model Transformations* (pp. 52-66). Springer Berlin Heidelberg

Iacob, M. E., Steen, M. W., Heerink, L., 2008. Reusable model transformation patterns. *In Enterprise Distributed Object Computing Conference Workshops*, *2008 12th* (pp. 1-10). IEEE.

Iovino, L., Pierantonio, A., Malavolta, I., 2012. On the Impact Significance of Metamodel Evolution in MDE. *Journal of Object Technology 11(3): 3: 1-33.*

Joault, F., 2005. Loosely Coupled Traceability for ATL. *In Proceedings of the European Conference on Model Driven Architecture workshop on traceability.* ECMDA.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., 2008. ATL: A model transformation tool. *Sci. Comput. Program. 72(1-2): 31-39*.

Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., & Wimmer, M., 2012. Model transformation by-example: a survey of the first wave. *In Conceptual Modelling and Its Theoretical Foundations* (pp. 197-215). Springer Berlin Heidelberg.

Kessentini, M., Sahraoui, H., Boukadoum, M., Omar, O. B., 2012. Search-based model transformation by example. *Software & Systems Modeling,* 11(2), 209-226

Kolovos, D. S., Paige, R. F., Polack, F. A., 2008. The epsilon transformation language. *In Theory and practice of model transformations* (pp. 46-60). Springer Berlin Heidelberg.

Langer, P., Wimmer, M., Kappel, G., 2010. Model-to-model transformations by demonstration. *In Theory and Practice of Model Transformations* (pp. 153-167). Springer Berlin Heidelberg.

Levy, F., Muniz, P., 2013. Applying MTBE Manually: a Method and an Example. MDEBE@MoDELS.

Matragkas, N., Kolovos, D., Paige, R., Zolotas, A., 2013. A Traceability-Driven Approach to Model Transformation Testing. AMT@MoDELS.

Mellor, S. J., 2004. MDA distilled: principles of model-driven architecture. Addison-Wesley Professional.

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

Object Management Group (OMG), 2009. Modeling and Analysis of Real-time and Embedded systems (MARTE), Version 1.0, http://www.omg.org/spec/MARTE/1.0/.

Object Management Group (OMG), 2011. Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.1.

Open Architecture Ware (oAW), 2010. XPand language reference.*http://www.openarchitectureware.org/pub/documentation/4.0/r20_xPandReference.pdf.*

Strommer, M., Wimmer, M., 2008. A framework for model transformation by-example: Concepts and tool support. *In Objects, Components, Models and Patterns* (pp. 372-391). TOOLS.Springer Berlin Heidelberg.

Sun, Y., Gray, J., 2013. End-User support for debugging demonstration-based model transformation execution. *In Modelling Foundations and Applications* (pp. 86-100). Springer Berlin Heidelberg.

Sun, Y., Gray, J., Delamare, R., Baudry, B., White, J., 2013. Automating the maintenance of nonfunctional system properties using demonstration-based model transformation. *Journal of Software: Evolution and Process 25(12): 1335-1356.*

Sun, Y., White, J., Gray, J., 2009. Model transformation by demonstration. *In Model Driven Engineering Languages and Systems* (pp. 712-726). Springer Berlin Heidelberg.

Toulmé, A., 2006. Presentation of EMF Compare Utility. *Eclipse Modeling Sympossium*.
Van Deursen, A., Visser, E., Warmer, J., 2007. Model-driven software evolution: A research agenda. *Proceedings of Int. Workshop on Model-Driven Software Evolution (MoDSE)*, ECSMR'07.
Varró, D. (2006). Model transformation by example. In Model Driven Engineering Languages and Systems (pp. 410-424). Springer Berlin Heidelberg.
Wagelaar, D., Van Der Straeten, R., Deridder, D., 2009. Module superimposition: a composition technique for rule-based model transformation languages. *Software and Systems Modeling* 9(3), 285-309.
Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W., 2012. Fact or Fiction– Reuse in Rule-Based Model-to-Model Transformation Languages. *In Theory and Practice of Model Transformations* (pp. 280-295). Springer Berlin Heidelberg.
Wimmer, M., Perez, S. M., Jouault, F., Cabot, J., 2012. A Catalogue of Refactorings for Model-to-Model *Transformations. Journal of Object Technology*, 11(2), 2-1.