Integrating a Testing Technique into the RTL Tool

Nguyen-Hoang Pham, Ngoc-Huyen Luong, Thi-Hanh Nguyen, Duc-Hanh Dang*

VNU University of Engineering and Technology

Abstract

Model transformation is an indispensable part in model-driven engineering. Such an importance has created a demand for transformation testing strategies and tools. This paper introduces an approach for testing model transformations using classifying terms. Classifying terms enable users to easily exercise equivalence class partitioning in order to validate model transformations, by giving them fine-grained control over the selection of input data and test oracles. By integrating this technique into an existing framework, we provide a way to validate model transformations using a graphical interface, in which the generation of input models and validation of output models are handled automatically.

Keywords: Model Transformation, Restricted Graph Transformation Language

1. Introduction

Model transformation, as one of the essential parts of Model Driven Engineering (MDE), is becoming more and more widely used for different objectives. In MDE, models which are generated from model transformations, are key artefacts of software projects. Thus, the quality of models depends on the quality of model transformations.

Testing is an effective way to validate model transformations. It allows detecting most common errors related to specification and implementation of model transformations. Model transformation testing has faced two main challenges that are the generation of test models and oracle functions [8, 13].

Firstly, testing model transformation generally tries to automate the generation of test cases. Test models are complex structures with data and behavior, which must conform to constraints defined in the source meta-model. Generating realistic test models automatically and efficiently is nontrivial. Secondly, a major challenge in model transformations testing concerns test oracles. The oracle procedure requires the comparison between the generated target model and the expected output model determined in a given test case. This task can be done either syntactically or semantically. Syntactically, the comparison algorithm must compare two graphs, and the task is often highly complex. Therefore, checking the semantics of a target model against pre-existing sources of knowledge given as constraints, e.g., post-conditions of transformations or invariants on the output language, is a more common method.

Additionally, model transformations deal with models, which are graph-based, therefore, the complexity of the testing process is much higher than code testing, especially when the model transformation involves a large number of classes and associations. Testing model transformation manually is therefore very time-consuming and error-prone. This has led to the need for techniques and tools to support testing model transformation automatically.

In this paper, we propose an approach for testing bidirectional model transformations using the classifying term concept as introduced in [4]. We also develop a graphical tool to realize our approach and to automate the two phases of testing, test case generation and oracle function generation, in a fast and reliable method. This tool is integrated into the existing

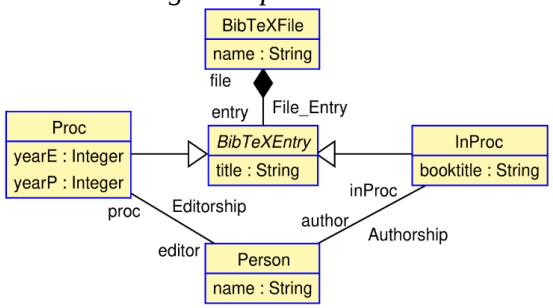
* Corresponding author. Email: hanhdd@vnu.edu.vn

transformation framework RTL [1] on the UML-based Specification Environment (USE).

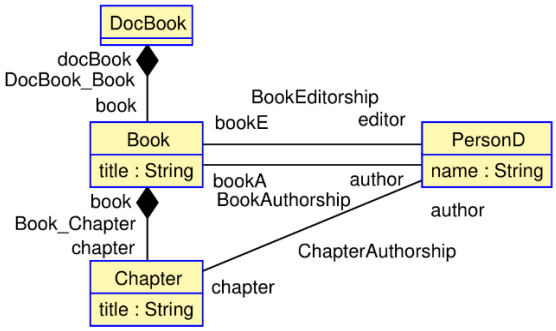
The rest of this paper is organised as follows. Section 2 presents the background of our work: Restricted Graph Transformation Language (RTL) and Classifying Terms (CTs). Section 3 introduces our proposal about testing transformations. Then, a support tool for black-box testing technique realizing our approach is presented in Sect. 4. Section 5 presents experimental results and discussions. Section 6 comments on related work. The paper is closed with conclusions and future work.

2. Preliminaries

2.1. A Running Example



a) BixTeX file meta-model



b) DocBook file meta-model

Figure 1. The meta-models of transformation

To better demonstrate our approach we consider the BibTeX2DocBook transformation that converts the information about proceedings of conferences in BibTeX format into the corresponding information encoded in DocBook format. Although two formats are different, they can be used for the same purpose – to store bibliography documents. Converting between the two formats is necessary when, for example, a user wants to import a BibTeX bibliography into his existing work in DocBook or vice versa. This example is a simple version of the BibTeX2DocBook model transformation example in ATL Transformation Zoo created by Eclipse [3] and is used to demonstrate a testing technique in [7]. The source and target meta-models used for the transformation are shown in Fig. 1.

This transformation will perform the following tasks:

- Transforming a BibTeXFile to a DocBook file and vice versa
- Transforming each Proc of the BibTeXFile to one Book with the same title in the equivalence DocBook file and vice versa.
- Transforming each Inproc in the BibTeXFile to one Chapter with the same title and belong to the Book whose title is the same as the Inproc booktitle and vice versa.
- Transforming each Person to one PersonD with the same name and role (author/editor) and vice versa

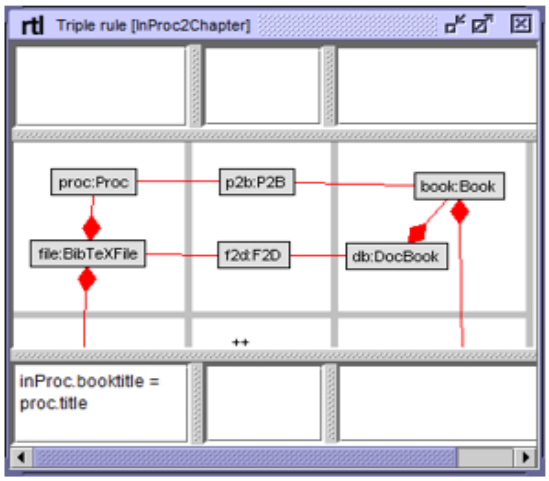
2.2. Implementing the transformation in RTL

There are many different languages for model transformations. Some aims at transforming models in one direction, like the ATL transformation language [8]. Others allow transforming models bi-directionally.

One of the approaches toward bidirectional model transformation is by using Triple Graph Grammar (TGG), a transformation language that is highly suitable for specifying bidirectional transformations [5]. In TGG model transformation specifications, the source models and target models are connected via correspondence model that represents the relationship and/or constraints between source and target model and the constraints between source elements and target elements in matching patterns. Pattern matching is a central concept in TGG.

In order to produce a target model from a source model, a transformation engine must perform pattern matching to find objects and links that match the *left-hand side* of a TGG rule, then create additional elements to match the TGG rule's *right-hand side*. Certain characteristics of triple graph grammar, such as its solid mathematical foundation and graph-based concept, enable users to specify model transformation rules in a declarative, high-level and even graphical manner.

```
rule InProc2Chapter
checkSource(
    file: BibTeXFile
    proc: Proc
    (file, proc): File_Entry
){
    inProc: InProc
    (file, inProc): File_Entry
    [inProc.booktitle = proc.title]
}checkTarget(
    db: DocBook
    book: Book
    (db, book): DocBook_Book
){
    chap: Chapter
    (book, chap): Book_Chapter
}checkCorr(
    (file, db) as (f, d) in f2d:F2D
    (proc, book) as (p, b) in p2b:P2B
){
    (inProc, chap) as (i, c) in i2c:I2C
    I2C:[self.c.title = self.i.title]
}end
```



(a) Declarative RTL rule *(b) Graphical RTL rule*

Figure 2. A TGG rule to convert a Proc object to a Book object and its visualisation in USE.

Restricted Graph Transformation Language (RTL) is a language and framework for bidirectional model transformation using TGG integrated with OCL (Object Constraint Language) constraints. The TGG rules are used to specify the transformation, while OCL defines the constraints on the source and target models and constraints on source and target elements in form of pre/post-condition, and invariants.

In order to specify the transformation, the user has to define a set of rules describing how to transform an element of the source model into its equivalence in the target model. Figure 2 demonstrates the RTL language syntax for a transformation rule. Thus, a transformation specification includes the source and target meta-models and a set of RTL rules presenting mapping patterns between source elements and target elements. The transformation specification with RTL can be automatically transformed into USE command to implement the transformation [1].

However, there are cases when a source model is used as input of the transformation program. The transformation program translates the source model into a target model, but the result of the transformation is incorrect. Figure 3 demonstrates the result when performing a model transformation on a BibTeX model (on the left), which consists of the equivalent DocBook model (on the right), and a correlation model (objects and links representing the mapping between source and target elements). It can be observed that the object PersonD1 has the ChapterAuthorship relationship with both Chapter1 and Chapter2; while its corresponding object, Person1, is only the author of InProc1. This is an unwanted behaviour which points to a mistake related to the transformation program or the specification.

Thus, additional validation measures must be introduced in order to ensure the transformation's correctness. In model transformation, the input is usually a set of source models, and validation is done by comparing the produced target models with expected ones. However, there are cases when only certain classes of source models reveal specification faults. Classifying terms, the technique underlying our implementation, can help users define custom model classes and generate input models that cover all classes.

2.2. Classifying terms

One of the difficulties in validating model transformation is how to select effective test cases among the infinite object model space. A large number of object models having the same

properties and behaving the same in the transformation might be taken into account. This would make the validation more time-consuming and might not guarantee to cover all possible scenarios.

To deal with this problem, we employ classifying terms introduced in [4] to divide the input model space into a finite set of equivalence classes. Classifying terms are OCL expressions which can be applied to a class model to calculate a characteristic value for each object model. The characteristic values can be either integral or Boolean, which will decide the number of equivalence classes each classifying terms can define. We would like to use multiple Boolean classifying terms, each defines a single piece of the classifying requirement, and then combine those classifying terms to get our desired equivalence classes.

```
oneProc
BibTeXEntry.allInstances->
  selectByType(Proc)->size() = 1
authorXorEditor
PersonB.allInstances->forAll(p|
  p.proc->isEmpty() xor
    p.inProc->isEmpty())
```

To demonstrate the use of classifying terms more clearly, we would like to give an example of two simple classifying terms being defined on the source meta-model of the BibTeX2DocBook transformation.

The first classifying term divides the input space into two equivalence classes. In the first equivalence class, BibTeXFile can only have one proceeding. The classifying term in this case will have the value True. While in the second equivalence class, BibTeXFile can have more than one proceeding (BibTeXFile cannot have zero proceedings as constrained by the meta-model invariant). The classifying term in this case will have the value False.

Similarly, the second classifying term focuses on the characteristic of whether a person can be both an editor of a proceeding and an author of a paper (InProc). These two terms in combination will divide the test input space into 4 equivalence classes. Each equivalence class can satisfy or not satisfy one or more classifying terms.

By using the USE plugin ModelValidator [9], which uses SAT solvers to generate different object models, in combination with these classifying terms, we can get one representative object model from each equivalence class.

Figure 4 demonstrates an object model corresponding to one solution of the resulting object models with the values of given

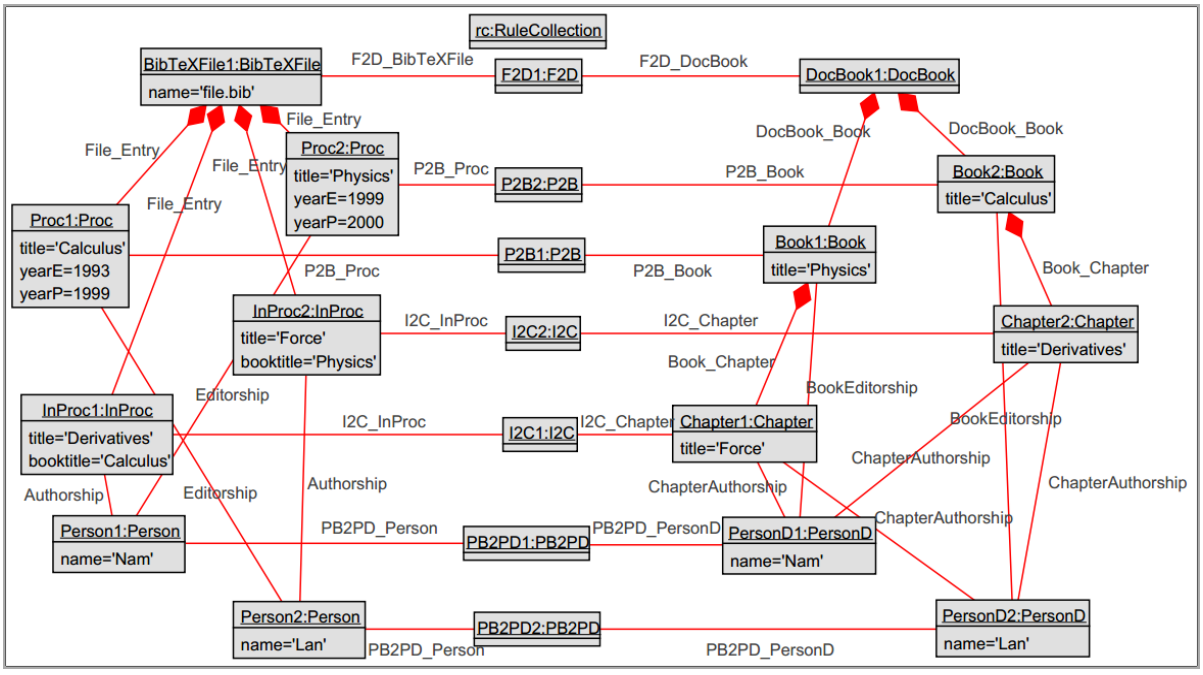
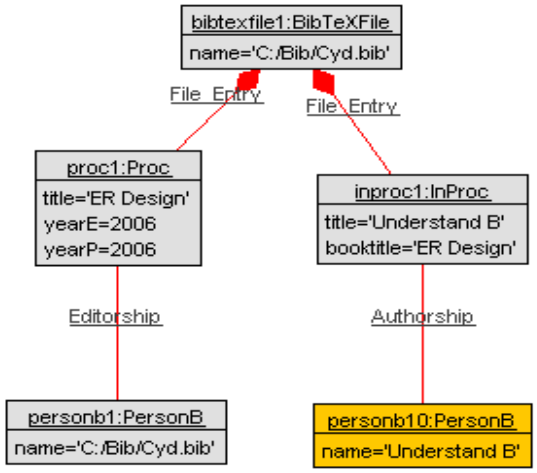


Figure 3. Example of a model produced by a faulty transformation

classifying terms ([oneProc] = true and [authorXorEditor] = true).

The using of classifying terms for testing model transformation in our approach are introduced in the following section.



[oneProc] = True, [authorXorEditor] = True

Figure 4. Object model generated by Model Validator.

3. Approach overview

In this paper, we provide an automatic validation method that integrates an existing transformation framework, RTL in [1].

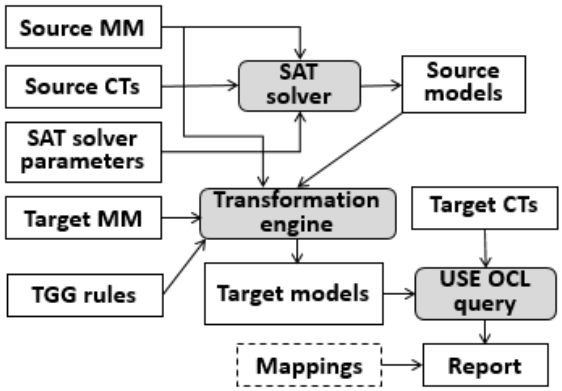


Figure 5. Overview of our approach

Figure 5 presents a high-level overview of the proposed approach. The source meta-model, classifying terms and a set of parameters are used as input to the SAT solver to generate a set of source models. The transformation engine takes the source and target meta-models, the RTL transformation rules as well as the source object models to produces a set of corresponding target models.

The validation is achieved by using two set of classifying terms. One set of classifying terms is applied to the source meta-model to partition the input space into equivalence classes and generate corresponding input models. The other set of classifying terms is applied to the corresponding transformed models. These classifying terms will define the properties that the output models are expected to possess.

Each input model, with its distinctive properties, when going through the transformation, will result in an output model with certain properties. By mapping the equivalence classes of the source and target models, we can determine if the model transformation behaves in the way as expected.

4. Support tool

In this section, we present our implementation of model validation using classifying terms in RTL. The framework is available as a plugin for USE (UML Specification Environment [10]).

There are several reasons for this choice. First, USE has support for the Object Constraint Language, which can be used to specify model invariants and classifying terms. Second, it possesses an extensible plugin system – as a result, its functionality has been enhanced with several plugins, including RTL and ModelValidator. Moreover, classifying term handling is incorporated into ModelValidator, greatly simplifying the development process. USE's graphical user interface enables users to create, visualize and edit models easily and interactively.

The result of the implementation, we developed a support tool for our approach. In following subsections, the input, output and workflow of the tool are presented.

4.1. Input

The support tool takes as input the source and target meta-models, transformation rules, ModelValidator configuration files, and source/target classifying terms; all of which are plain text files.

The source and target models are UML models specified in USE's syntax. They are represented as class diagrams in USE. The ability to show and hide model features (e.g.,

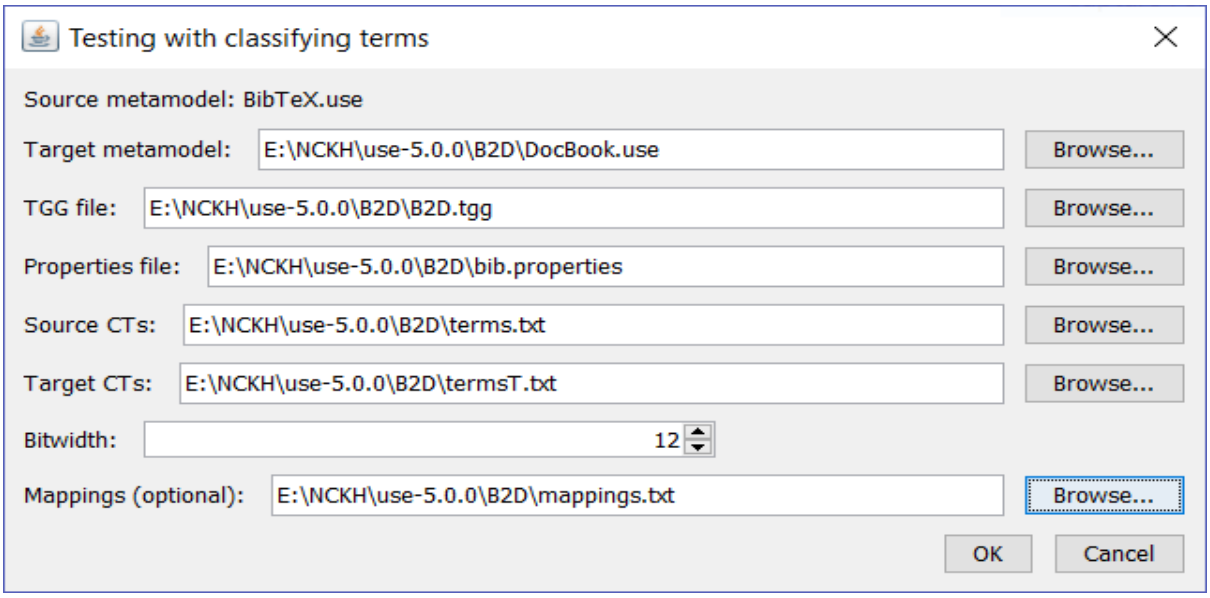


Figure 6. The input specification dialogue box.

operations, role names, association names) is supported, as well as printing and PDF exporting.

Transformation rules are triple graph grammar rules written in the RTL language. The RTL plugin provides the ability to visualise them using object diagrams. The transformation direction (forward or backward) is specified in the rule file as a keyword.

In order to limit the search space, a configuration file is required to restrict size and domain information of object models. This file can be created manually or with the help of a GUI included in the plugin; and options such as the number of objects/links and possible values for each attribute are configurable.

In our implementation, each classifying term is defined in two consecutive lines in a text file – the former contains the name of the term while the latter is the OCL query associated with it.

Finally, an optional mapping file can be provided. It contains a list of patterns in the format of `sourceCTs -> targetCTs`, in which each side specifies a list of Integer/Boolean values. Support for negative patterns (by appending the '!' character) and wildcards (represented by the '*' character) is also provided.

4.2. Workflow

After the input artefacts have been created, the user provides them to the RTL plugin by means of a dialogue box, shown in Fig. 6. The plugin performs three steps to show the result of validating model transformation with the given classifying terms.

The first step is test case generation. The plugin first parses and validates the transformation-related artefacts (i.e., meta-models and TGG rules). The classifying terms are then checked for syntactical correctness. The source classifying terms (in the case of forward transformation) or the target classifying terms (in the case of backward transformation) together with the configuration file and Bitwidth are used to configure ModelValidator's SAT solver. ModelValidator comes with several SAT solvers, which can be selected from USE's command line interface. Subsequently, the SAT solver is executed. For each combination of classifying term values, it tries to generate a model with the data provided in the configuration file. If an error occurs during the process or no valid models can be found, the process is halted and the user receives an error message.

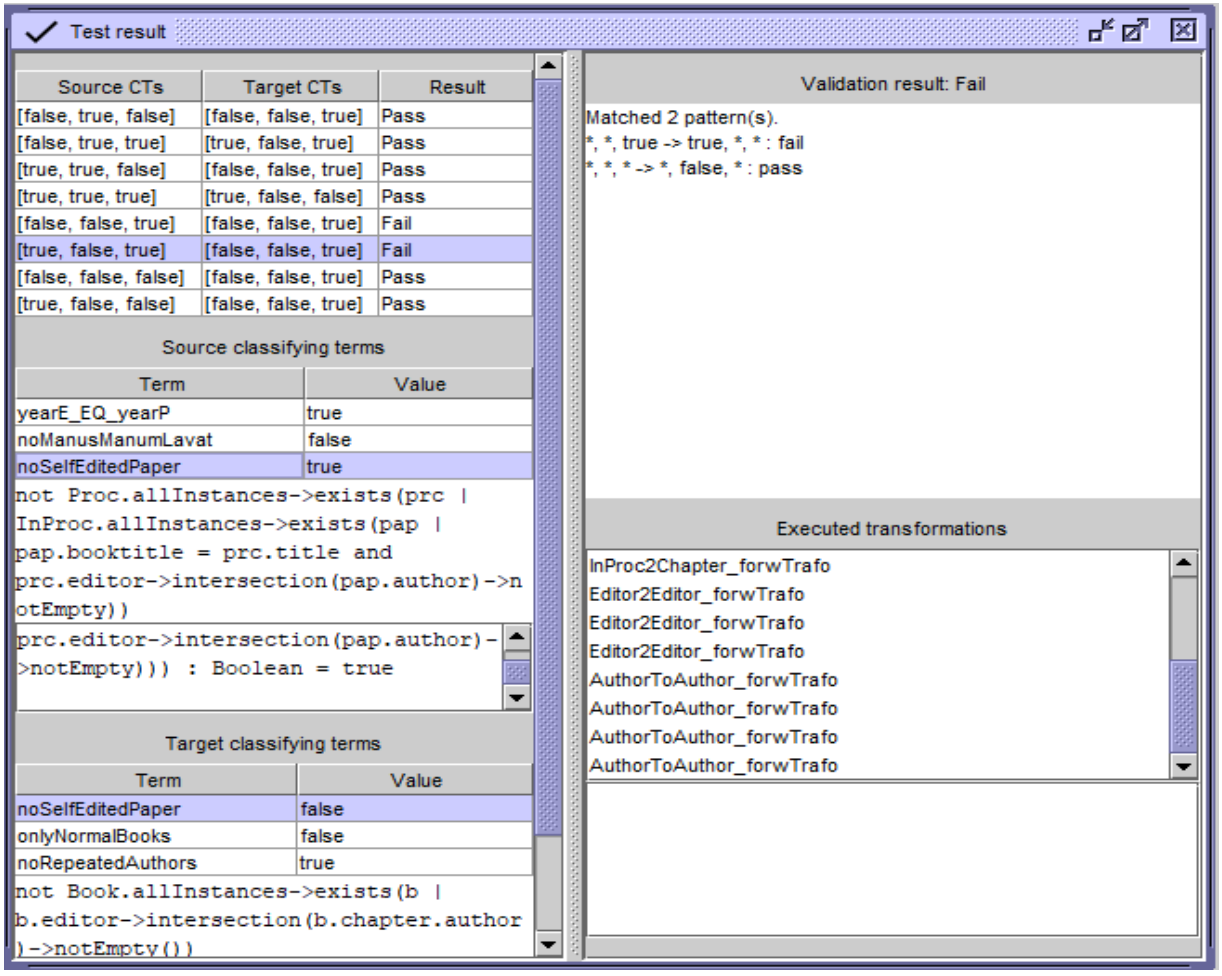


Figure 7. The validation result dialoque

In the second step, the RTL engine executes the transformation on the generated input model in order to generate a correspondence output model that conforms to the target meta-model.

The third step is oracle checking. The input and output models are checked against the mappings. The plugins will generate a report based on the result to show which test passes and which test fails. In our approach, we use the partial oracle checking using contracts on the target model after transformation. These contracts are constructed in target classifying terms that used to check whether the output models satisfy requirements come from users.

4.3. Output

If the process completes successfully, a report is displayed, as shown in Fig. 7, containing the values of the classifying terms for each source – target model pair. A number of additional outputs are provided to assist in the debugging process.

More specifically, selecting a classifying term brings up its associated OCL query and evaluation log; and selecting a model pair displays its validation result in detail, as well as the list of executed commands produced by the transformation engine.

The result can be visualised by selecting a model pair, in which case the state of the system is reflected in an object diagram, as in Fig. 8. When a specific transformation is selected, its affected objects (matched objects and created objects) are shown in a different colour.

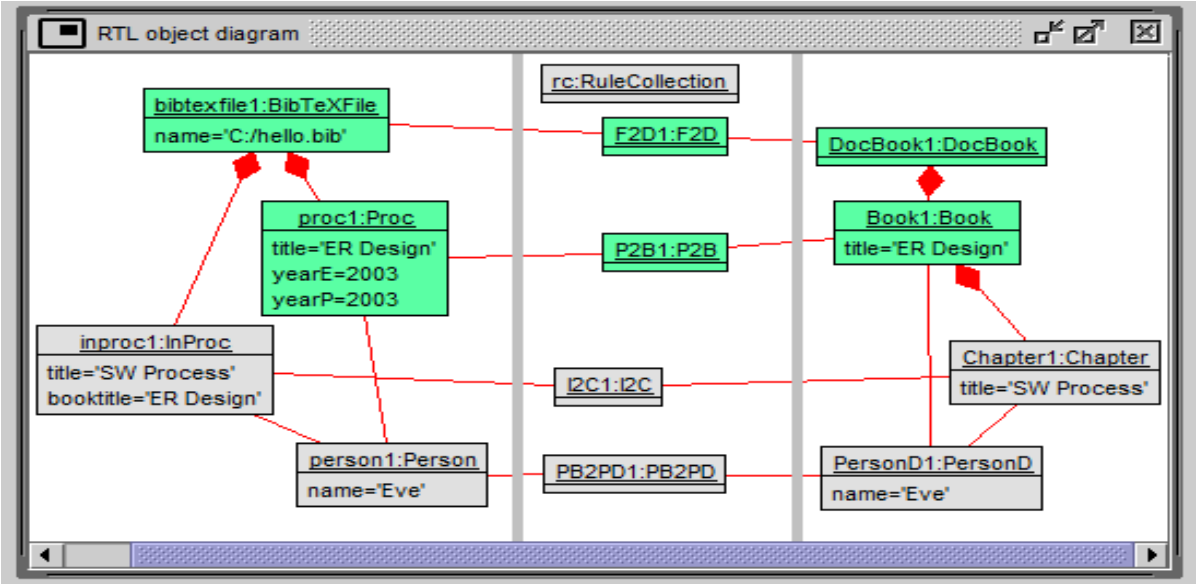


Figure 8. The selected model pair from the validation results shown in Fig. 7.

For each model pair, the values of the classifying terms are matched against the patterns in the mapping file. Only the patterns whose left side matches the source classifying term values take part in the validation process. If for all these patterns, the right side matches the target classifying term values as well, the model pair is considered to have passed the validation. On the other hand, the existence of a pattern whose right side does not match the target classifying term values suggests that the transformation specification does not meet the requirements.

5. Results and discussion

5.1. Experiments

In this section, we present some results when RTL is utilised in the development of TGG rules for the BibTeX to DocBook transformation.

For this experiment, we used the same classifying terms for source and target models as in [4], listed Fig. 9. Our main focus lies on the `noSelfEditedPaper` on both sides, whose value is true if no authors are also the editor of one of their own papers. It can be concluded that a correct transformation should produce target models having the same `noSelfEditedPaper` value as their corresponding source models.

On the other hand, on each modelling language ones are interested in different information, so they can define specific requirements on the source and target model by the different classifying terms.

Suppose that we want to concentrate on different characteristics of the input models of the BibTex2DocBook transformation. First, proceedings have two dates: the year of the conference event (yearE) and the year in which proceedings were published (yearP). This situation is expressed in the term "`yearE_EQ_yearP`". Second, we want to have some input models in which two editors of proceeding are not allowed to invite the other to have a paper there. This situation is presented in the term "`noManusMamumLavat`".

Besides, in output models of BibTeX2DocBook transformation we are interested in normal books, i.e. those which are not composition of papers selected by an editor; but instead all chapters are written by the same person, the book author. Also, books in which no author writes more than one paper could be of interest too. We define these situations in two classifying terms "`onlyNormalBooks`" and "`noRepeatedAuthors`", respectively.

For validating the model transformation BibTeX2DocBook, we experimented the transformation program using the set of above -

```
--source classifying terms
yearE_EQ_yearP
Proc.allInstances->forAll(yearE = yearP)
noManusManumLavat
not Person.allInstances->exists(p1, p2 | p1 <> p2 and p1.proc->exists(prc1 |
p2.proc->exists(prc2 | prc1 <> prc2 and
InProc.allInstances->select(booktitle = prc1.title)->exists(pap2 |
pap2.author->includes(p2) and InProc.allInstances->select(booktitle =
prc2.title)->exists(pap1 | pap1.author->includes(p1))))))
noSelfEditedPaper
not Proc.allInstances->exists(prc | InProc.allInstances->exists(pap |
pap.booktitle = prc.title and
prc.editor->intersection(pap.author)->notEmpty))
```

```
--target classifying term
noSelfEditedPaper
not Book.allInstances->exists(b |
b.editor->intersection(b.chapter.author)->notEmpty())
onlyNormalBooks
Book.allInstances->forAll(b | b.editor->isEmpty() and b.chapter->forAll(c |
c.author=b.author))
noRepeatedAuthors
Book.allInstances()->forAll(b | b.chapter->forAll( c1, c2 | c1 <> c2 implies
c1.author->intersection(c2.author)->isEmpty()))
```

Figure 9. Classifying terms for source models and target models

explained terms as in Fig. 9 and an appropriate configuration file as well as the Bitwidth set to 12. The validation task took approximately 9 seconds to complete, generating 8 pairs of source – target model pairs, each source model representing a different equivalence class defined by the source classifying terms.

Thanks to the automation of the process, modellers can have an overview of the results and easily spot discrepancies, resulting in faster error diagnosis. In our first run of the test, we observed that the `noSelfEditedPaper` term of the target model is `false` in some cases where the term `noSelfEditedPaper` of the source model has a value of true as shown in Fig. 7. This observation of unexpected behaviour helped us detect a missing condition in the transformation of the `Authorship` association, which created `ChapterAuthorship` links when the behaviour is not appropriate.

5.2. Discussion

Due to the fact that model transformation with TGG rules requires searching throughout the system to find candidates, the transformation step does not scale well when the input contains too many objects and links. In practice, since the validation is performed on models with just enough elements to exhibit certain characteristics of interest, this limitation is of little concern. ModelValidator's configuration file also helps limit the search space by defining the maximum number of objects and links.

ModelValidator's support for classifying term values is restricted to Boolean and Integer types, and ranges are not supported. However, classifying terms with other types of values can be rewritten to be compatible with the tool. For example, a classifying term specifying ranges of integers can be converted into multiple Boolean terms using comparison operators.

6. Related work

In the context of model transformation testing and validation, a number of approaches for generating and selecting input data have been put forward. Sen et al. [11] proposed a similar approach to that implemented in this paper, in which models are generated with Alloy from model fragments. However, the model fragments are automatically extracted from the meta-model, as opposed to written manually; thus, the user cannot specify the exact scenario in which he/she wants to validate

the transformation. A model generation tool called ASSL [12] is built into USE; while it can also be used to generate models, its imperative approach means that users have to learn a new language. ASSL can be used in conjunction with OCL invariants to replicate classifying terms, but the ASSL specification may have to be rewritten in order to comply with the invariants.

Oracle function is also a challenge in model transformation testing and validation [13]. In this paper, the mapping between source and target classifying terms are used as the oracle function. According to Mottu et al. [6], this approach belongs to the group of oracle functions using an OCL assertion. As mentioned in [6], other commonly-used approaches include *generic contracts*, which is already implemented in RTL's post-conditions [1]; and oracle functions that make use of graph comparisons, such as *model snippets/fragments* [11] and comparing the output with an expected output model, whether using graph [7] or textual comparison approaches as survey in [8].

7. Conclusion and future work

In this paper, we have pointed out how the technique of transformation validation using classifying terms can help speed up the software development process. We also discuss about how to choose requirements that are used to define classifying terms for different testing aims.

However, without a proper graphical, interactive environment, the approach can be hard to use. Therefore, we have incorporated this technique into USE – a modelling tool with visualisation capabilities and good support for UML/OCL. The technique as well as the implementation can be streamlined by introducing the automatic creation of model generation parameters and the automatic inference of classifying terms from meta-models. This is our aim for a future version of the tool.

Acknowledgment

We would like to thank Assoc. Prof. Dr. Pham Ngoc Hung, VNU University of Engineering and Technologies for his useful reviewing.

References

[1] D.-H. Dang and M. Gogolla, "An OCL-Based Framework for Model Transformations," VNU Journal of Science: Computer Science and Communication Engineering, vol. 32, no. 1, pp. 44-57, 2016.

[2] J. R. C. J. D. Gehan M. K. Selim, "Model transformation testing: the state of the art," 2012 .

[3] A. Kusel, J. Schönböck, M. Wimmer, W. Retschitzegger, W. Schwinger and G. Kappel, "Reality Check for Model Transformation Reuse: The ATL Transformation Zoo Case Study," AMT@ MoDELS, 2013.

[4] F. Hilken, M. Gogolla, L. Burgueño and A. Vallecillo, "Testing models and model transformations using classifying terms," Software & Systems Modeling, pp. 1-28, 2016.

[5] A. Schürr, "Specification of Graph Translators with Triple Graph Grammars," International Workshop on Graph-Theoretic Concepts in Computer Science, pp. 151-163, 1994.

[6] J.-M. Mottu, B. Baudry and Y. Le Traon, "Model transformation testing: oracle issue," ICST, 2008, pp. 105-112, 2008.

[7] J.-w. Ko, K.-y. Chung and J.-s. Han, "Model transformation verification using similarity and graph comparison algorithm," Multimedia tools and applications, vol. 74, no. 20, pp. 8907-8920, 2015.

[8] Lukman Ab. Rahim, Jon Whittle, "A survey of approaches for verifying model transformations", SoSyM, Volume 14, Issue 2, pp 1003–1028, 2013.

[9] M. Kuhlmann, L. Hamann and M. Gogolla, "Extensive validation of OCL models by integrating SAT solving into USE," TOOLS, pp. 290-306, 2011.

[10] M. Richters and M. Gogolla, "Validating UML models and OCL constraints," International Conference on the Unified Modeling Language, pp. 265-277, 2000.

[11] S. Sen, B. Baudry and J.-M. Mottu, "Automatic model generation strategies for model transformation testing,", ICMT, pp. 148-164, 2009.

[12] M. Gogolla, J. B. Bohling and M. Richters, "Validation of UML and OCL models by automatic snapshot generation," International Conference on the Unified Modeling Language, pp. 265-279, 2003.

[13] B. Baudry, T. Dinh-Trong, J.-M. Mottu, D. Simmonds, R. France, S. Gosh, F. Fleurey and Y. Le Traon, "Model transformation testing challenges," ECMDA, 2006.