Iterative Development of Transformation Models by Using Classifying Terms

Frank Hilken[1] Loli Burgueño[2] Martin Gogolla[1] Antonio Vallecillo[2]

[1]University of Bremen, Bremen, Germany
`{fhilken|gogolla}@informatik.uni-bremen.de`

[2]University of Málaga, Málaga, Spain
`{loli|av}@lcc.uma.es`

Abstract

In this paper we propose an iterative process for the correct specification of model transformations, i.e., for developing correct *transformation models*. This permits checking the correctness of a model transformation specification before any implementation is available. The proposal is based on the use of classifying terms for partitioning the input space and for simplifying the testing process.

1 Introduction

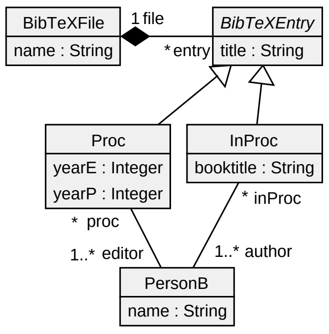
As model-driven engineering (MDE) is progressively being adopted by industry, model transformations (MT) are becoming commonplace. Their complexity is also growing, since the problems they need to deal with are increasingly harder. From simple structural migration, model queries or pattern-based code-generation, they now have to cope with complex model synthesis, behavioural analysis and stream data processing. This has led, among other things, to the need of *engineering* model transformations [7].

In this context, the specification of a model transformation becomes a critical task, in particular for checking the correctness of its implementations. Please note that correctness is not an absolute property, but needs to be checked against a contract, or specification, which determines the expected behaviour, the context whether such a behaviour needs to be guaranteed, as well as some other properties of interest. The specification states what should be done, but without determining how. The problem, again, is that the specification of a model transformation can be as complex as the transformation itself.

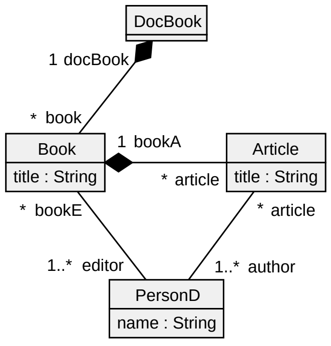
Here we will make use of the fact that, in essence, a model transformation is an algorithmic specification (either declarative or operational) of the relationship between two or more models, and more specifically, of the mapping from one model to another [12]. Thus, one way to specify a model transformation is by means of a *transformation model* that defines such a relation [3].

In this paper we propose an iterative process for the correct specification of model transformations, i.e., for developing correct transformation models. This is done by testing the relationship specified by the transformation. We use classifying terms [5] to implement a divide-and-conquer strategy that permits simplifying the analysis of the mappings established by the model transformation, by focusing on particular input models. This approach is based on the ideas proposed by *equivalence partitioning*, a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. Once a problem is detected, the specification is revised to solve it. This process is iterated until the specification is found free of errors.

One additional benefit of our approach, and a difference with regard to previous proposals, is that it permits testing the specification without needing an implementation of the model transformation, which is the common way in which model transformation specifications and implementations are tested (by checking one against the other, i.e., using a particular kind of *redundancy testing*). In our approach, model transformation specifications can be tested on their own, before any implementation is available.



(a) BibTEX Model



(b) DocBook Model

Figure 1: Separate initial models used as the base for the transformation model.

The structure of this paper is as follows: Section 2 presents the case study and describes the generation of test cases. The iterative process of the transformation model development is presented in Sect. 3. Section 4 compares our work to related topics of interest. Finally, Section 5 concludes the paper and presents future work.

2 Preliminaries

Here, we develop a transformation model using a test-driven approach. Assisted by a model validator that completes partial object diagrams to consistent system states, bugs in the model description are identified and the transformation model is iteratively fixed until it is complete, works as expected and no more bugs can be detected.

2.1 Running Example

The two initial models, the transformation is developed for, are simple bibliography managers shown in Fig. 1. One resembles the information of BibTEX files and the other resembles the DocBook format. At the top of the structure is the `BibTeXFile` or the `DocBook` with entries for proceedings and inproceedings that will be mapped to books and articles, respectively. Additionally, authors and editors are saved.

Differences in the models exist in the structure how the information is saved and the capabilities saved information. For instance, the BibTEX model saves the year of the event (`yearE`) and the year of publication (`yearP`) for proceedings, which are not present in the DocBook model. Therefore, the transformation model will not be able to cover all attributes of the models.

2.2 Characteristics of Classifying Terms

Classifying terms [5] are an instrument to explore model properties. They are employed in the model validation and verification process when one has a UML and OCL model given, and one has in addition a so-called configuration that specifies a finite search space connecting the models elements, basically classes, associations, attributes, and datatypes, with finite sets of possible populations. As an important validation task, the developer might want to see all object models that satisfy the UML and OCL class model and that fit to the configuration. Classifying terms give the developer an explicit option to formulate their understanding of two object models being different. Thus when the developer scrolls through all object models matching the UML class model, the OCL invariants and the configuration, they will only encounter models with different characteristics with respect to the classifying term.

The technical realization works as follows. The developer specifies a closed OCL query term, i.e., a term without free variables, that can be evaluated in an object model and that returns a characteristic value. This term is called ‘classifying term’. Each newly constructed object model has to show a different characteristic value for the classifying term. The classifying term determines an equivalence relationship on all object models. Two object models with the same characteristic value belong into the same equivalence class. Our approach decides to choose only one representative from each equivalence class.

The definition of the classifying terms corresponds to the mapping that the developer has in mind about the transformation, and how it should map source models of each source equivalence class into target models of the corresponding target class. In this sense, the source and target classifying terms should be ‘in correspondence’. Our approach can also help checking such expected mappings.

2.3 Test Case Generation with Classifying Terms

In order to efficiently test our transformation model, we create test cases using classifying terms. This technique results in test cases covering developer chosen equivalence classes that are determined by the classifying terms. As a result, exactly one test case is automatically generated for each equivalence class. They have a much broader coverage than only few manual tests. Depending on a clever choice for the classifying terms, the advantages are faster test execution, due to a smaller test suite, as well as a higher error detection rate per test case that cover many, if not all, errors already.

We defined three classifying terms for the source metamodel and three for the target metamodel of our running example. They infer eight equivalence classes in each one. The source classifying terms focus on different characteristics of the input models of the transformation. First, we want to have input models in which proceedings have different conference event and publication years, and other in which they coincide. Second, we want to have some sample input models in which two editors of proceedings invite the other to have a paper there; respectively, we also want to have input models in which this "manus-manum-lavat" situation does not happen. Finally, we want to have some source models with proceedings edited by one of the authors of the papers in the proceedings, and other input models with no "self-edited" proceedings:

```
[ 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))
```

The classifying terms for the target models identify properties of interest in the target space, such as self-edited books, or books which are authored by a single person. They are not shown here for space reasons.

3 Developing a Transformation Model Assisted by Completions

The process of the transformation model development is outlined in Fig. 2. A first revision is manually created based on the specification of the transformation. Using the test cases generated by classifying terms, a verification tool simulates the transformation by completing the transformation model. The results are checked against the specification and errors are fixed by refining the model until the transformation complies with the specification.

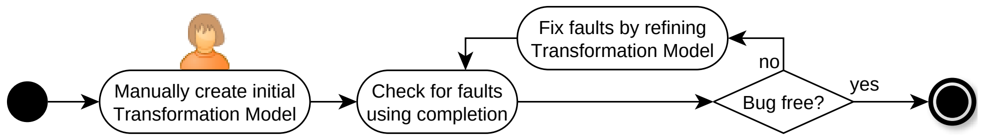

Manually create initial Transformation Model
Check for faults using completion
Fix faults by refining Transformation Model
no
Bug free?
yes

Figure 2: Overview of the iterative development process.

3.1 Initial Transformation Model

We gave the metamodels to an experienced developer with knowledge in OCL and the description in natural language of the transformation model. Basically, each BibTeXFile should have a direct correspondence with a DocBook; each Proc with a Book; each InProc with an Article, and each PersonB with a PersonD. Moreover, all the relationships between the source objects are kept in the target model but there exists a new one between a book and an article when the corresponding InProc has as booktitle the corresponding Proc. With this, the developer provided the following four constraints (one for every pair of objects) as the transformation specification.

```
context TM inv BibTeX2DocBook:
   BibTeXFile.allInstances->forAll(file | DocBook.allInstances->exists(dB |
      file.entry->selectByType(Proc)->forAll( proc | dB.book->one(b | proc.title = b.title))))
```

```
context TM inv Proc2Book:
   Proc.allInstances->forAll(proc | Book.allInstances->exists(book | proc.title = book.title and
      proc.editor->forAll(editorP | book.editor->exists(editorB | editorP.name = editorB.name and
         book.article->forAll(art | InProc.allInstances->one(inP | inP.booktitle = art.title))))))

context TM inv InProc2Article:
   InProc.allInstances->forAll(inP | Article.allInstances->exists(art | inP.title = art.title and
   art.bookA.title = inP.booktitle and inP.author->forAll(authP |
      art.author->exists(authA | authP.name = authA.name))))

context TM inv PersonB2PersonD:
   PersonB.allInstances->size() = PersonD.allInstances->size() and
   PersonB.allInstances->forAll( p | PersonD.allInstances->exists(pd | p.name = pd.name))
```

3.2 Contradictory Constraints

Given the transformation model and test cases obtained using the classifying terms previously defined in Sect. 2.3, the model validator could not create a valid system state from any of the test cases. There is no possible model that fulfils all the conditions. The reason is a fault in the constraint `Proc2Book`. At the end of it, Proceeding titles are compared to Article titles, which are different layers of information and thus contradictory. This contradiction can be revealed using the counterproof that the model validator provides when such contradiction occurs. The constraint `Proc2Book` looks as follows after the modification[1].

```
context TM inv Proc2Book:
   Proc.allInstances->forAll(proc | Book.allInstances->exists(book | proc.title = book.title and
      proc.editor->forAll(editorP | book.editor->exists(editorB | editorP.name = editorB.name and
         book.article->forAll(art | InProc.allInstances->one(inP | inP.title = art.title))))))
```

3.3 Checking the Relation Established by the Transformation

Now the model validator is able to complete the source object diagrams according to the specification given in the transformation model. Figure 3 shows inside the square the third source object diagram obtained with the classifying terms and, outside the square, the completion the model validator generated.

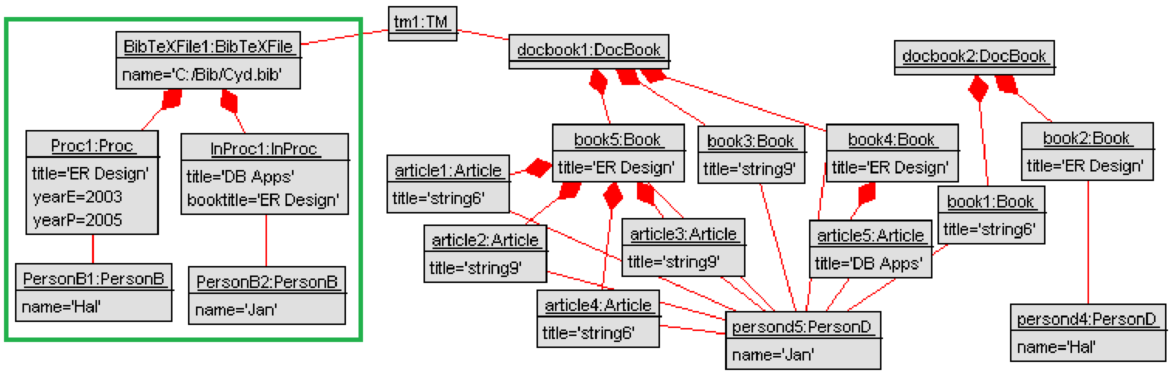
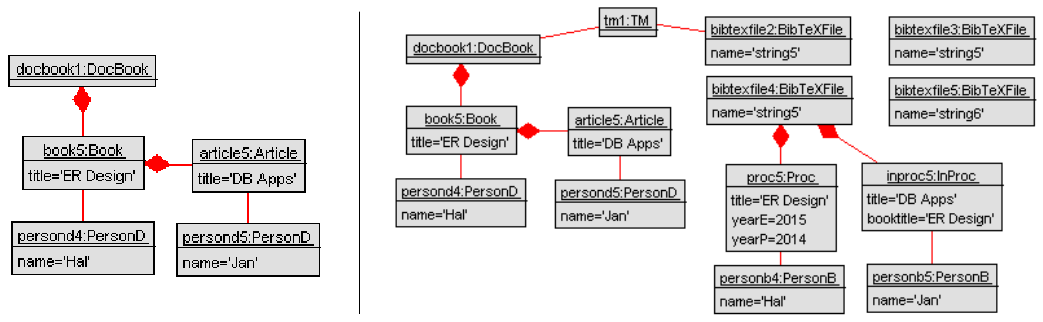


Figure 3: Completion for the third BibTeX model.

Several problems can be easily detected. First of all, there is no one-to-one correspondence between the source and target objects: the target contains more objects than it should. This does not match with the expected behaviour of the transformation, which should establish a one-to-one relation between the source and target model elements. We can make use of the fact that at the specification level, the relationship established by the transformation can be read and traversed in both ways, i.e., *the transformation model is direction-neutral*. Then, we manually created the expected target object diagram (left-hand side of Fig. 4) and completed it. The result after the completion is shown in the right-hand side of Fig. 4, where we can see that the source model obtained is not the third object diagram obtained applying the classifying terms (the objects inside the square in Fig. 3), which means that the transformation model, as it is, does not establish the expected relationship between the source and target model elements. This problem is solved by adding object equality to the constraints to limit the number of objects generated when completing the object diagrams.

[1]The changes made in the specification are highlighted with a gray background.



(a) Expected DocBook model for the third BibTeX model.

(b) Completion for the DocBook model.

Figure 4: Proof of non-bidirectionality.

Another observable problem is that the names were supposed to be unique — i.e., a DocBook should not have two Books with the same name and a Book should not have two Articles with the same name — but they are not. In the constraints the `one` operation, which is intended for the uniqueness, is placed inside the body of an `exists` operation, which leads to the situations shown in Fig. 3. In the object diagram there are two DocBooks but only one of them respects the uniqueness (`docBook2`). As there exists one DocBook object fulfilling the constraint, the overall system state is valid. In order to achieve the desired behaviour, the `exists` expressions need to be replaced by `one` expressions. After the two changes the constraints look like the following.

```
context TM inv BibTeX2DocBook:
   BibTeXFile.allInstances->size() = DocBook.allInstances->size() and
   BibTeXFile.allInstances->forAll(file | DocBook.allInstances->one(dB |
      file.entry->selectByType(Proc)->forAll( proc | dB.book->one(b | proc.title = b.title))))

context TM inv Proc2Book:
   Proc.allInstances->size() = Book.allInstances->size() and
   Proc.allInstances->forAll(proc | Book.allInstances->one(book | proc.title = book.title and
      proc.editor->size() = book.editor->size() and
      proc.editor->forAll(editorP | book.editor->one(editorB | editorP.name = editorB.name and
         book.article->forAll(art | InProc.allInstances->one(inP | inP.title = art.title)))))

context TM inv InProc2Article:
   InProc.allInstances->size() = Article.allInstances->size() and
   InProc.allInstances->forAll(inP | Article.allInstances->one(art | inP.title = art.title and
      inP.author->size() = art.author->size() and art.bookA.title = inP.booktitle and
      inP.author->forAll(authP | art.author->one(authA | authP.name = authA.name))))

context TM inv PersonB2PersonD:
   PersonB.allInstances->size() = PersonD.allInstances->size() and
   PersonB.allInstances->forAll( p | PersonD.allInstances->one(pd | p.name = pd.name))
```

4 Related Work

In the field of Model-Driven Engineering, model transformations are key elements and even the simplest ones may contain faults as pointed in [11]. Therefore, their testing, validation and verification have been subject of investigations. In this work, we see model transformations as transformation models and, as a software artifact, there exists the need of checking their correctness as well.

Several lines of work consider the testing of model transformation implementations. Some of them are dynamic approaches that execute the model transformations given an input model or a set of them. References [9] and [13] present a contribution for debugging model transformations. The work in [1] analyses the trace model in order to find errors. Also, Tracts [12] are a static black-box approach that establishes contracts between the source and target metamodels which define the transformation specification. They have been combined with classifying terms in [5] for automatically constructing relevant source model test cases. In addition to Tracts, other static approaches have been proposed [8] that allow the specification of contracts in a visual manner, and the work in [4] that looks at the differences between the actual output model generated by the transformation and the expected output model.

Reference [2] proposes a dynamic testing technique defining equivalence classes for the source models in a similar manner as it is done with classifying terms. Nevertheless, their proposal is less expressive as they do not consider the use of OCL, less flexible and lacks full automation. Reference [6] presents a mechanism for generating test cases by analysing the OCL expressions in the source metamodel in order to partition the input model space. This is a systematic approach similar to ours, but focusing on the original source model constraints. Our proposal allows the developer partitioning the source (and target) model space independently from these constraints, in a more flexible manner.

Finally, the work in [10] proved the correctness of specifications by making use of algebras. Our approach can be seen as a first step and as an easier and cheaper way that does not require for the developer to have any extra knowledge or create any other software artifact.

5 Conclusion and Future Work

In this paper we have presented an iterative approach for the correct development of *transformation models*. These models provide the specifications of model transformations, and with our proposal they can be checked before any implementation is available, and independently from any of them.

There are several lines of work that we plan to address next. In the first place, we would like to validate our proposal with more transformations, in order to gain a better understanding of its advantages and limitations; identify different contexts of use in which our approach works well and other in which the results are not satisfactory (and why), and build a repository of thoroughly tested and validated transformation models that can be reused by the community. Second, we plan to improve the tool support to further automate all tests, so human intervention is kept to the minimum. Finally, we need to define a systematic approach of defining classifying term and transformation model testing using the preliminary ideas outlined in this paper.

Acknowledgment We thank the reviewers for the many constructive ideas and detailed feedback. This work was partially funded by the German Research Foundation (DFG) under grant GO 454/19-1 and by the Spanish Research Projects TIN2011-23795 and TIN2014-52034-R.

References

[1] V. Aranega, J.-M. Mottu, A. Etien, and J.-L. Dekeyser. Traceability mechanism for error localization in model transformation. In *Proc. of ICSOFT'09*, 2009.

[2] B. Baudry, T. Dinh-Trong, J. Mottu, D. Simmonds, R. France, S. Ghosh, F. Fleurey, and Y. Le Traon. Model transformation testing challenges. In *ECMDA Workshop on Integration of MDD and Model Driven Testing*, 2006.

[3] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow. Model transformations? transformation models! In *Proc. of MODELS'06*, volume 4199 of *LNCS*, pages 440–453. Springer, 2006.

[4] A. García-Domínguez, D. S. Kolovos, L. M. Rose, R. F. Paige, and I. Medina-Bulo. EUnit: A Unit Testing Framework for Model Management Tasks. In *Proc of MODELS'11*, LNCS, pages 395–409. Springer, 2011.

[5] M. Gogolla, A. Vallecillo, L. Burgueño, and F. Hilken. Employing classifying terms for testing model transformations. In *Proc. of MODELS'15*, LNCS. Springer, 2015.

[6] C. A. González and J. Cabot. Test Data Generation for Model Transformations Combining Partition and Constraint Analysis. In *Proc. of ICMT'14*, volume 8568 of *LNCS*, pages 25–41. Springer, 2014.

[7] E. Guerra, J. de Lara, D. Kolovos, R. Paige, and O. dos Santos. Engineering model transformations with transML. *Software & Systems Modeling*, 12(3):555–577, 2013.

[8] E. Guerra, J. de Lara, M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger. Automated verification of model transformations based on visual contracts. *Autom. Softw. Eng.*, 20(1):5–46, 2013.

[9] M. Hibberd, M. Lawley, and K. Raymond. Forensic debugging of model transformations. In *Proc. of MODELS'07*, volume 4735 of *LNCS*, pages 589–604. Springer, 2007.

[10] F. Orejas and M. Wirsing. On the specification and verification of model transformations. In *Semantics and Algebraic Specification, Essays Dedicated to Peter D. Mosses on the Occasion of his 60th Birthday*, volume 5700 of *LNCS*, pages 140–161. Springer, 2009.

[11] J. Sánchez Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL model transformations. In *Proc. of MODELS'15*, LNCS. Springer, 2015.

[12] A. Vallecillo, M. Gogolla, L. Burgueño, M. Wimmer, and L. Hamann. Formal specification and testing of model transformations. In *Formal Methods for Model-Driven Engineering (SFM)*. Springer, 2012.

[13] M. Wimmer, G. Kappel, J. Schönböck, A. Kusel, W. Retschitzegger, and W. Schwinger. A Petri Net based debugging environment for QVT Relations. In *Proc. of ASE'09*, 2009.