TractsTool: Testing Model Transformations based on Contracts

Loli Burgueño[1], Manuel Wimmer[2], Javier Troya[1], and Antonio Vallecillo[1]

[1] GISUM/Atenea Research Group. Universidad de Málaga, Spain
[2] Business Informatics Group, Vienna University of Technology, Austria

Abstract. Model transformations play an important role in Model-Driven Engineering (MDE), and as their size and complexity grow, there is an increasing need to count on tool support for testing their correctness. In this work, we present *TractsTool*, a tool for specifying and testing several different kinds of model transformations, e.g., model-to-model, model-to-text, and text-to-model transformations, based on contracts.

Keywords: Model transformation, contract-based design, testing, tool support

1 Introduction

The basis of the Model-Driven Engineering (MDE) paradigm is the usage of models. They have become key artifacts from which a system may be (semi-)automatically derived and implemented. Model transformations are the key mechanism of MDE for manipulating models. Model-to-model (M2M) transformations take as input models conforming to specific metamodels and transform them into other models conforming to other (or the same) metamodels. In addition to M2M transformations, model-to-text (M2T) and text-to-model (T2M) transformations are of major importance. The former are frequently used to produce code from models, while the latter are typically used for reverse engineering of legacy systems or injecting artifacts to the MDE technical space.

As the size and complexity of model transformations grow, there is an increasing need to count on mechanisms and tools for testing their correctness [1]. One option is to validate the behavior of the transformation and its associated properties (termination, determinism, etc.) using formal methods and their associated tools (cf. e.g., [2,3]). The main drawback is that the whole transformation needs to be dealt with, what makes this approach inefficient for large models and transformations. An alternative approach [4,5] consists of not validating the transformation for the full input space, but trying to certify that it works for a selected set of test input models. This may not fully prove correctness, but it can be very useful for identifying bugs in a cost-effective and light-weight manner.

The tool presented in this paper follows the latter approach. We generalize *model transformation contracts* [6] for the specification of the properties that need to be checked for a transformation, and subsequently, apply the ASSL language [7] to generate input test models. As presented in [8], our approach can be applied to M2M as well as to M2T and T2M transformations.

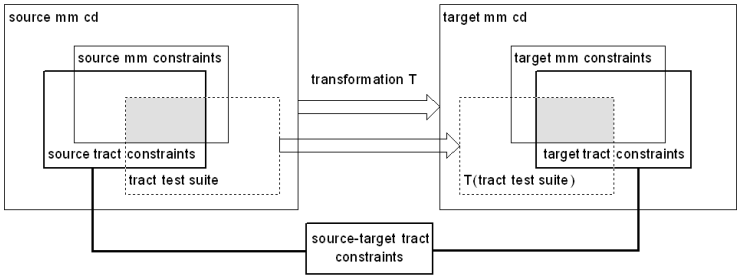

Fig. 1. Building Blocks of a Tract

After this introduction, we have a glance at the foundations of the Tracts approach in Section 2. In Section 3 we describe how we consider to test M2T and T2M transformations, whereas in Section 4 we elaborate on how TractsTool is realized. We close the paper by presenting the conclusions and future work in Section 5.

2 Tracts at a glance

In Fig. 1 we show the main ingredients of the Tracts approach [9]: a source and target metamodel, the transformation T under test, and a transformation contract, from here on and for short called Tracts, which consists of a tract test suite and a set of tract constraints. A test suite contains a set of models on which constraints are going to be checked. In the figure, the test suite and its transformation result are shown with dashed lines. The different Tracts are depicted with thick lines. The user can specify five different types of constraints: the source and target class diagrams are restricted by source and target metamodels constraints, and the Tracts additionally impose source (SRC), target (TRG), and source-target (SRC/TRG) tract constraints. All these constraints are expressed by means of invariants formulated in the Object Constraint Language (OCL) [10].

For exemplifying how Tracts are defined, we make use of the well-known Class-to-Relational transformation example[3]. In the following listing, we present for each constraint type, i.e., source, target, and source-target constraint, one example that the Class-to-Relational transformation should fulfill. For reasons of brevity, we only present some very basic constraints. For more complex ones the interested reader is referred to our project website[4].

```
-- SRC: Classes have to have unique names
  Class.allInstances -> unique(c|c.name)
-- SRC/TRG: For each class, a table has to be produced
  Class.allInstances -> forall(c|Table.allInstances -> exists(t|t.name = c.name))
-- TRG: The columns of a table have to have unique names
  Table.allInstances -> forall(t|t.columns ->unique(c|c.name))
```

[3] http://www.eclipse.org/atl/atlTransformations

[4] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/Tracts

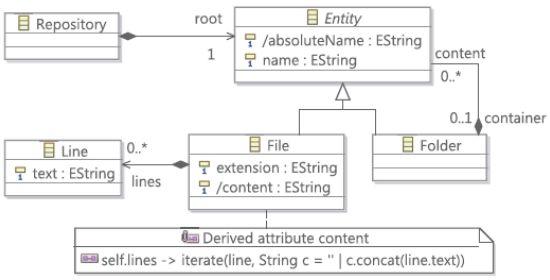

Fig. 2. Metamodel for representing text artifacts and repositories

3 Testing M2T and T2M transformations

Compared to testing M2M transformations, M2T and T2M transformations have not gained much attention when it comes to testing. In our approach, we adopt current techniques for testing M2M transformations to the problem of testing T2M and M2T transformations. The prerequisite of using existing M2M transformation techniques is to have metamodels for the input and output of the transformations. However, for the side that is dealing with just text of M2T and T2M transformations, no metamodels are usually available. Even more problematic, when considering T2M and M2T transformations, a set of metamodels and T2M parsers may be required as a prerequisite to reach the MDE technical space. To alleviate the burden from T2M and M2T transformation engineers, we employ a generic approach that may be used for any transformation task where text is involved as input or output of the transformations. The main mechanism we employ is to represent text within a generic metamodel in order to transform M2T and T2M transformation specification problems into equivalent M2M transformation specification problems [8].

Artifacts involved in M2T and T2M transformations are normally organized in a hierarchical folder structure. For instance, the output of a M2T transformation may not be just a single file but several, which should be also arranged in a certain folder structure forming a repository of artifacts. Fig. 2 shows the metamodel for representing text artifacts stored in repositories. Meta-class `Repository` represents the entry point to the root folder containing folders and files. While folders just contain a name, files have in addition an extension as well as a content. The content of files is represented by lines that are sequentially ordered. By having this generic metamodel and an accompanying generic T2M parser, we are able to inject text artifacts into models and reuse model-based tools such as the TractsTool. Going back to the Class-to-Relational transformation example, assume that we are now using a M2T transformation for directly producing Data Definition Language (DDL) code. The following Tract may be defined for this scenario. For more complex Tracts the interested reader is again referred to our project website.

```
-- SRC/TRG: For each class, a CREATE TABLE statement has to be produced
 File.allInstances -> exists(f| f.name = 'db_schema' and Class.allInstances ->
   forAll(c|f.lines -> exists(l|l.text.matchesRE('^CREATE␣TABLE␣' + c.name))))
```

4 Realization of TractsTool

Fig. 3 shows, in terms of a UML activity diagram, the general workflow of TractsTool. Please note that we assume for T2M/M2T transformations that the input/output has been transformed into a model as discussed in the previous section.

Our tool needs as input: (i) source and target metamodels in EMF format, (ii) input models in EMF format or ASSL program for producing input models, (iii) ATL transformation to produce the target models or expected target models in EMF format, and (iv) the set of constraints to be checked. To check the constraints and eventually report their violations, the source and target models and their respective metamodels are transformed to USE [11], a fully-fledged UML/OCL environment. For this, we identified the corresponding modeling features of USE for the modeling features of EMF. When dealing with EMF, Ecore is used as metamodeling language, which is equivalent to the core of the UML class diagram language offered by USE. For representing models, UML object diagrams are a natural choice when moving to USE. Thus, we constructed explicit Ecore-based metamodels for the modeling languages of USE. Then, by using M2M transformations, we are able to obtain USE models from EMF models that are subsequently serialized to the USE format by M2T transformations. In addition, we also developed the transformations for the inverse direction, i.e., transforming USE models to EMF models. This is needed for cases where ASSL programs are used for producing source models (in USE format) and ATL is used for producing the target models from source models (in EMF format). Please note that for formulating and checking the constraints, the source and target (meta-)models are internally merged in the USE representations.

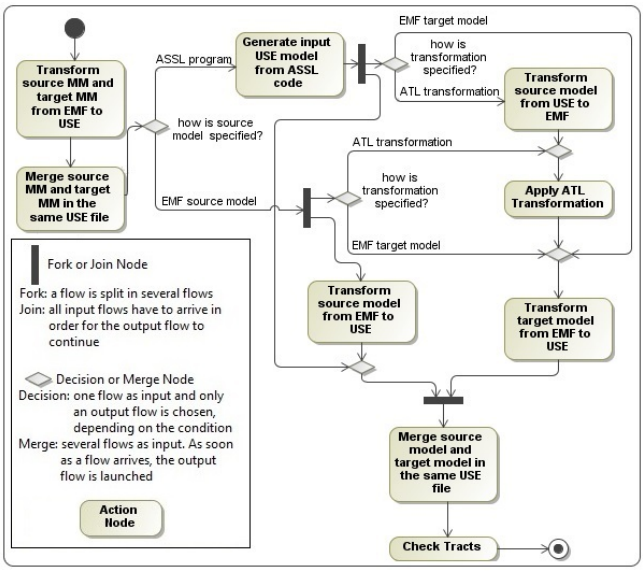

Fig. 3. Transformation chain used in TractsTool

5 Conclusions and future work

We have presented the foundations of TractsTool, a tool for light-weight verification of any kind of model transformation by following the Tracts approach.

There are several lines of research that are still subject to ongoing work. Firstly, we plan to investigate how the Knowledge Discovery Metamodel (KDM) may be used for defining contracts that are programming language independent and reusable for a family of code generators. Secondly, the TractsTool's internal transformations presented in Section 4 are time-consuming when transforming very large models. Another objective is to provide a mechanism to generate automatically the ASSL program that in turn generates a set of default input models so that the user is not required to develop such programs by hand. Another improvement that may be made is to change the information that is displayed after the verification process. Specifically, diagnostic models are an interesting line of research in this respect. Finally, we plan to include error tracking [12] directly in TractsTool. Thus, in case a Tract fails, the transformation engineer should be supported in identifying the transformation rule(s) that have caused the failure.

Acknowledgements. This work is partially funded by Research Project TIN2011-23795.

References

1. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In: Proc. of the 1st Workshop on Verification of Model Transformations (VOLT) @ ICST, IEEE (2012) 921–928
2. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Verification and validation of declarative model-to-model transformations through invariants. JSS **83**(2) (2010) 283–302
3. Troya, J., Vallecillo, A.: A Rewriting Logic Semantics for ATL. JOT **10** (2011) 5:1–29
4. Brottier, E., Fleurey, F., Steel, J., Baudry, B., Traon, Y.L.: Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In: Proc. of the 17th Int. Symposium on Software Reliability Engineering (ISSRE), IEEE (2006) 85–94
5. Fleurey, F., Baudry, B., Muller, P.A., Traon, Y.L.: Qualifying input test data for model transformations. SoSym **8**(2) (2009) 185–203
6. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. ECEASST **24** (2009)
7. Gogolla, M., Bohling, J., Richters, M.: Validating UML and OCL Models in USE by Automatic Snapshot Generation. SoSym **4**(4) (2005) 386–398
8. Wimmer, M., Burgueño, L.: Testing M2T/T2M Transformations. In: Proc. of the 16th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS). Volume 8107 of LNCS., Springer (2013) 203–219
9. Gogolla, M., Vallecillo, A.: Tractable Model Transformation Testing. In: Proc. of the 7th European Conf. on Modeling Foundations and Applications (ECMFA). Volume 6698 of LNCS., Springer (2011) 221–236
10. Cabot, J., Gogolla, M.: Object Constraint Language (OCL): the Definitive Guide. In: Formal Methods for Model-Driven Engineering. Volume 7320 of LNCS., Springer (2012) 58–90
11. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. SCP **69**(1-3) (2007) 27–34
12. Burgueño, L., Wimmer, M., Vallecillo, A.: Towards Tracking *Guilty* Transformation Rules. In: Proc. of the 1st Workshop on the Analysis of Model Transformations (AMT) @ MODELS, ACM (2012) 27–32