Towards Analysing Non-Determinism in Bidirectional Transformations [⋆]

Romina Eramo, Romeo Marinelli, Alfonso Pierantonio, and Gianni Rosa

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica
Università degli Studi dell'Aquila, Italy
`name.surname@univaq.it`

Abstract. In Model-Driven Engineering, the potential advantages of using bidirectional transformations are largely recognized. Despite its crucial function, bidirectionality has somewhat limited success also because of the ambivalence concerning non-bijectivity. In fact, in certain situations more than one admissible solution is in principle possible, despite most of the current languages generate only one model at time, possibly not the desired one.
In this paper, we propose to manage non-determinism during the design process. The approach aims to analyze bidirectional transformations with the purpose to detect ambiguities and support designers in solving non-determinism in their specification.

1 Introduction

In Model-Driven Engineering (MDE) [18] bidirectionality in transformations has been always regarded as a key mechanism [20]. Its employment comprises mapping models to other models to focus on particular features of a system, simulate/validate a given application, and primarily keeping a set of interrelated models synchronized or in a consistent state. Despite its relevance, bidirectionality has rarely produced anticipated benefits as demonstrated by the lack of a language comparable to what ATL[1] represents for unidirectional transformations. Among the reasons why bidirectional techniques had limited success there is the ambivalence concerning non-bijectivity: when bidirectional transformations are non-bijective, there may be multiple *update policies* to transform two models into a consistent state, introducing uncertainty and non-determinism [8].

Most current languages are able to generate only one model. Thus, non-injective transformations involved in round-tripping can give rise to results, which are somewhat unpredictable. In these cases, the solution is normally identified according to heuristics, language implementation decisions and/or to the order the rules are written, as it happens with Medini [14] and ModelMorf [15]. Recently, few declarative approaches [6, 12, 4] to bidirectionality have been proposed; they are able to cope with the non-bijectivity by generating all the admissible solutions of a transformation at once. Among them, the Janus Transformation Language [6] (JTL) is a model transformation language specifically tailored to support bidirectionality and change propagation. Its semantics relies on Answer Set Programming (ASP) [9] in order to generate multiple

[⋆] This research was supported by the EU through the Model-Based Social Learning for Public Administrations (Learn Pad) FP7 STREP project (619583).

[1] http://www.eclipse.org/atl/

models. However, the JTL transformation engine is not able to detect non-deterministic rules, i.e., rules that can give place to multiple solutions, at static-time.

There have been several works analyzing semantic issues and idiosyncrasies of bidirectional model transformations. [21] emphasizes the need of a clear semantics to grant developers full control about what the transformation does. The author goes even further by claiming that a transformation must be deterministic in order to ensure that developers will find the transformation behavior predictable. However, there exists cases in which designers are not able to disambiguate their transformations. In fact, they may lack the information needed beforehand to take appropriate design decisions. Moreover, very often the non-determinism in a transformation becomes evident only after its execution, especially if the language implementation resolves latent ambiguity by means of default strategies whose behavior is unclear to developers.

In this paper, we present an approach to statically detect non-determinism in bidirectional transformations, i.e., without executing them. To this end, Answer Set Programming (ASP) [9] is exploited to realize a logic environment for the analysis of bidirectional transformations. In particular, transformations are translated into logical rules and constraints able to analyze the behavior of the transformation with an emphasis on non-determinism. The approach aims to support designers in solving non-determinism in their specifications by detecting the rules that give place to alternative solutions at design time[2].

The paper is organized as follows. Section 2 introduces the problem by means of an example. Section 3 describes the static analysis of bidirectional transformation. Sections 4 and 5 present the proposed approach and show it in practice. Section 6 describes related work. Finally, Section 7 draws some conclusion and future work.

2 A motivating example

Non-determinism is a frequent aspect which affects model transformation design and implementation. In this section, we describe how *ambiguous* mappings in a model transformation may cause multiplicity in the generated solution, i.e., to somewhat a form of uncertainty.

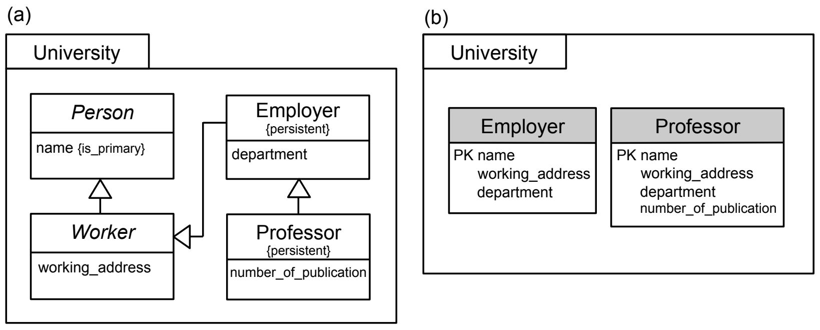


Fig. 1: The UML model (a) and the correspondent RDBMS model (b)

Scenario Let us considering a typical round-trip problem based on a non-bijective class diagram to relational data base (*UML2RDBMS*) benchmark scenario [7, 10]. Only

[2] Note that, the proposed analysis environment is general as does not depend on the JTL engine

persistent classes are mapped to correspondent tables and their attributes to columns in the tables, including inherited attributes. In order to preserve all the information of the source diagram, attributes of non-persistent classes have to be distributed over those tables stemming from persistent classes which access non-persistent ones.

The UML model in Fig. 1(a) shows the package `university` that is composed of an inheritance hierarchy of classes, whereas the correspondent schema is depicted in Fig. 1(b). Let us suppose that the generated model is manually modified (e.g., for satisfying new requirements) as depicted in Fig. 2(a): a new column `email` has been added in the table `employer`. This gives place to an interesting situation since such modifications can be reflected to the source model in Fig. 1(a) in three alternative ways: the attribute (corresponding to the manually added column) can be a member of the class `employer` or with each of parent classes `worker` and `person`, as in Fig. 2(b). Consequently, although not shown in the figure, starting from each of this alternatives, every subclass will inherit the attribute. Thus, more than one source model propagating the changes exist.

Implementation The *UML2RDBMS* bidirectional transformation, which relates class diagrams and relational database models, has been implemented by means of the Janus Transformation Language (JTL) [6].

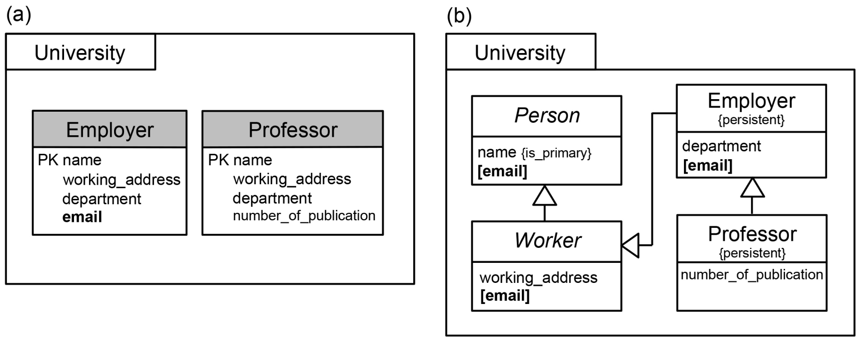


Fig. 2: The modified RDBMS model (a) and the correspondent UML model (b)

In Listing 1.1 we report a fragment of the transformation which is expressed in the textual concrete syntax of JTL and applied on models given by means of their Ecore representation within the EMF framework[3]. In particular, the following relations are defined: *a) Class2Table*, which relates classes and tables in the two different metamodels, *b) Attribute2Column*, which relates attibutes and columns in the two different metamodels and *c) SuperAttribute2Column*, which relates attributes of the parent class to columns of the corresponding child table. The *when* and *where* clause specify conditions on the relation. In particular, the when clause in Line `18` allow to navigate the parent classes of each attribute, then the where clause in Line `19` generates the correspondent columns. These relations are bidirectional, in fact both the contained domains are specified with the construct *enforce*.

The forward application of the transformation is illustrated in Fig. 1, where the UML model is mapped to the correspondent RDBMS model. As aforementioned the

[3] http://www.eclipse.org/modeling/emf/

transformation is non-injective. The back propagation of the changes showed in Fig. 2 gives place to the following situation: the transformation execution produces a set of three models each one represents a different solution where the column `email` belongs to each of the three different classes of the hierarchy. In particular, the modified RDBMS target model in Fig. 2(a) is mapped back to the UML source models in Fig. 2(b).

Such non-determinism can be resolved by specifying in the transformation that any new column of the table must be mapped to the corresponding attribute of the class from which the table is generated (for instance, the new column `email` of the table `employer` is mapped to the attribute `email` of the class `employer`).

```
transformation UML2RDBMS(uml:UML, rdbms:RDBMS)  { ...
 top relation Class2Table {
  cn, an: String;
  enforce domain uml c:Class {
   is_persistent = true,
   name = cn,
   attrs = attr: Attribute { name = an}
  };
  enforce domain rdbms t:Table {
   name = cn,
   cols = col: Column { name = an }
  };
  when { ... }
  where { Attribute2Column(c, t); }
 }
 relation Attribute2Column {
  an, at : String;
  enforce domain uml c:Class {
   attrs = attr: Attribute { name = an, owner = c, is_primary = false }
  };
  enforce domain rdbms t:Table {
   cols = col: Column { name = an, owner = t }
  };
  when { ... }
 }
 top relation SuperAttributeToColumn{
  enforce domain uml c: Class {
   parent = sc: Class { }
  };
  enforce domain rdbms t: Table { };
  when { ClassToTable(c,t) or (cc = c.parentOf and SuperAttributeToColumn(cc,t));}
  where { AttributeToColumn(sc, t); }
 } ...
```

Listing 1.1: A fragment of the UML2RDBMS transformation in JTL

Thus, the considered piece of transformation can be made deterministic by accommodating the mentioned refinement. However, when size and complexity of metamodels and transformations grow, the designer should be supported by an automated tool capable of detecting the ”guilty” rules in the transformations.

3 Static analysis of bidirectional transformations

Typically, bidirectional transformations are specified by a collection of rules which define mapping from source to target metamodel elements, and viceversa. By considering existing relational languages [16, 6], mappings among metamodel elements are expressed as relations that are executed in both the directions. In this context, transformations are models as well, therefore amenable to model operations.

Analysis of model transformations can be used for a wide range of assessments including the satisfaction of specific requirements, verifications/validation, and performance analysis. Existing approaches and tools involve testing (e.g., test case generation for model transformations), or analysis performed on executing programs (e.g., run-time monitoring) [1]. In contrast, static analysis is performed without actually executing the model transformation or generating test cases; in fact, it is performed on some (abstract) version of the transformation code with the scope to allow designer to understand and review the code at design-time.

Non-determinism is a critical aspect in bidirectionality. Thus, detecting the ambiguous fragments of a transformation at design-time may prevent unwanted behaviors like a severe increase of the solution space. At this scope, it is of paramount relevance to support the designer in understanding which rule is ambiguous and how to refactor the transformation in order to reduce non-determinism. This is intrinsically difficult as it not unusual that the backward (forward) execution of an seemingly deterministic forward (backward) transformation generates more than one alternative. The challenge consists of precisely identifying those combinations of rules which are responsible of the solution multiplicity.

The analysis is performed by considering not only the behavior of individual statements and declarations, but rather including the complete transformation. Information obtained from the analysis are used for detecting possible coding ambiguities. In particular, the approach aims to detect so-called *non-deterministic portions* of the transformation represented by collections of rules that if executed together may cause multiple alternative solutions. Please note how the degree of non-determinism of a single fragment depends on the model instance given as input.

In this paper, we propose to use the Answer Set Programming (ASP) [9] to analyze bidirectional transformation. ASP is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems and based on the stable model (answer set) semantics of logic programming. Then the ASP solver[4] finds and generates, in a single execution, all the possible models which are consistent with the logic rules by a deductive process. The proposed ASP-based engine is able to analyze the model transformation specification and verify non-determinism conditions expressed by means of rules and constraints. The environment has been implemented as a set of plug-ins of the Eclipse framework and mainly exploits EMF[5] as illustrated in the next section.

4 Description of the approach

Figure 3 illustrates the approach, which is comprised of two main steps. The first step translates a model transformation specification into a notation suitable for the analysis. The second analyzes the information extracted in the previous step and produces a feedback. These two steps are explained in more details in the following.

(1) Translation of the transformation into the analysis notation

A model transformation consists of a set of rules which define mapping among element types of the involved left- and right-metamodels. Generally, a rule is constrained with pre- and post-conditions that limit its applicability and behavior. This step defines a

[4] http://www.dlvsystem.com/

[5] http://www.eclipse.org/modeling/emf/

characterization that permits to capture only the information which is relevant for performing the analysis (neglecting irrelevant details which are not pertinent to our purpose). The notation used for representing the elicited information is the *Transformation Analysis Language* (see Fig. 3). Each rule is represented by means of two sets containing the left- and right-patterns, respectively. Whereas, each pattern is represented by a class and values for any of its properties and references. As defined in OCL[6], a pattern can be viewed as a set of variables, and a set of constraints that model elements bound to those variables must satisfy to qualify as a valid binding of the pattern. In other words, a pattern can be considered a template for objects and their properties that must be located in a candidate model to satisfy the rule. The model representing the bidirectional model transformation (*Bidirectional Transformation Model* in Fig. 3) must be translated into the corrisponding model for the analysis (*Transformation Analysis Model*) by means of a semantic anchoring [5] able to translate rules and constraints as set of patterns.

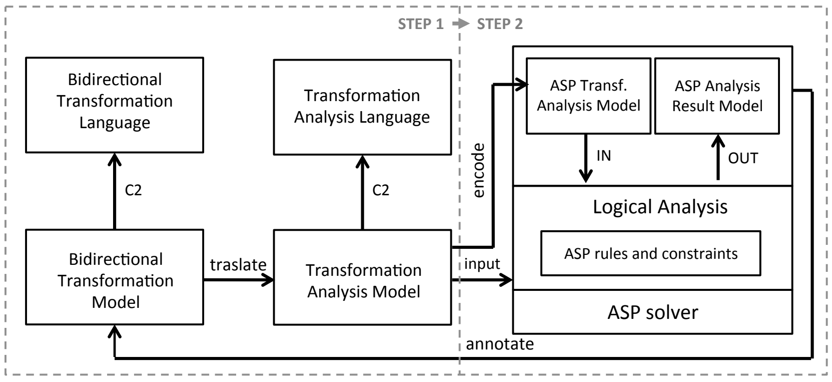

STEP 1
STEP 2
Bidirectional Transformation Language
Transformation Analysis Language
C2
C2
Bidirectional Transformation Model
traslate
Transformation Analysis Model
encode
input
annotate
ASP Transf. Analysis Model
ASP Analysis Result Model
IN
OUT
Logical Analysis
ASP rules and constraints
ASP solver

Fig. 3: Architecture overview of the approach

(2) ASP-based analysis

Once the analysis models are generated by abstracting from the concrete transformations, they need to be translated into ASP in order to be analyzed. The analysis models and the metamodels involved in the transformations are consistently consistently encoded as a knowledge base, whereas the properties which are required to be checked are translated in constraints and verified by ASP rules. As aforementioned the scope of the analysis is to provide an automated detection of non-deterministic rules. In particular, it generate sets of logically related rules whose simultaneous applications may generate multiple admissible solutions. When the ambiguous rules are marked by the analysis process, the designer can refactor the initial specification by resolving the non-determinism. In Listing 1.2 a fragment of the ASP rules for the detection of non-deterministic code is presented. In particular, the rule in Lines `1-4` deduces pairs of *equal domains* by comparing patterns and predicates of each domain. The rule in Lines `6-9` deduces *non-deterministic relations*; in particular, for each transformation direction, relations with equal domains are detected. Finally, the rule in Lines `11-13`

[6] http://http://www.omg.org/spec/OCL/

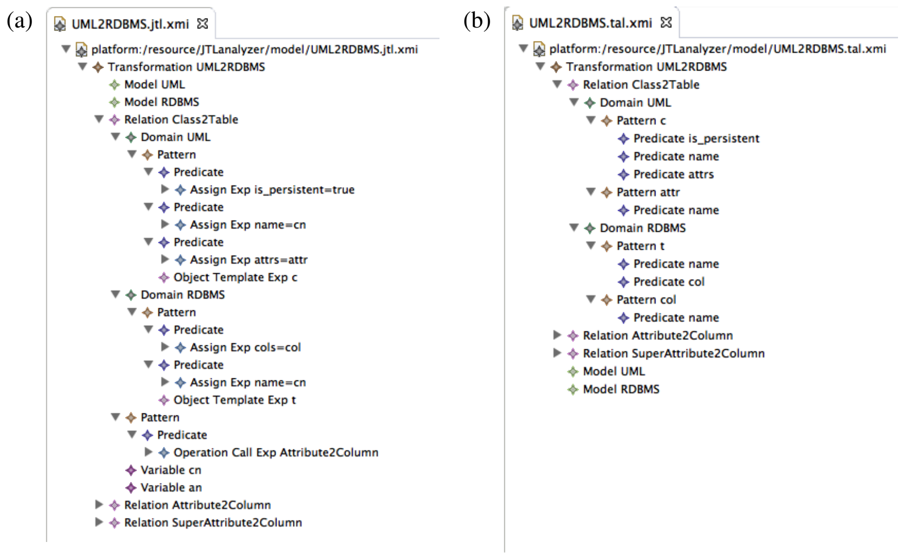
deduces pair of *ambiguous relations* which cause non-determinism if simultaneously executed.

```
are_equal_domains(ID1, ID2, Dom) :-
 have_equal_patterns(ID1, ID2, Dom, MC, MCName),
 have_equal_predicates(ID1, ID2, Dom, MC, MCName),
 not have_different_patterns(ID1, ID2, Dom).

non_deterministic_relation(ID, RelName, DomLeft) :-
 are_equal_domains(ID, ID2, DomRight),
 relation_name(ID, RelName),
 tranformation_model(DomRight), tranformation_model(DomLeft), DomRight != DomLeft.

are_ambiguous_relations(ID1, ID2, DomLeft) :-
 are_equal_domains(ID1, ID2, DomRight),
 tranformation_model(DomRight), tranformation_model(DomLeft), DomRight != DomLef
```

Listing 1.2: A fragment of the ASP rules and constraints for the analysis

5 Running the approach

In this section, we present an application of the proposed approach to the UML2RDBMS example presented in Sect. 2. The goal is to illustrate how to use of the approach in practice by exploiting the developed environment. In particular, we analyze the JTL implementation of the UML2RDBMS transformation[7]. First, the bidirectional transformation is translated into the analysis model. Then, the analysis is executed to highlight those rules which may be responsible of non-determinism.


(a)

```
UML2RDBMS.jtl.xmi
▼ platform:/resource/JTLanalyzer/model/UML2RDBMS.jtl.xmi
  ▼ Transformation UML2RDBMS
      Model UML
      Model RDBMS
    ▼ Relation Class2Table
      ▼ Domain UML
        ▼ Pattern
          ▼ Predicate
            ▶ Assign Exp is_persistent=true
          ▼ Predicate
            ▶ Assign Exp name=cn
          ▼ Predicate
            ▶ Assign Exp attrs=attr
            Object Template Exp c
      ▼ Domain RDBMS
        ▼ Pattern
          ▼ Predicate
            ▶ Assign Exp cols=col
          ▼ Predicate
            ▶ Assign Exp name=cn
            Object Template Exp t
      ▼ Pattern
        ▼ Predicate
          ▶ Operation Call Exp Attribute2Column
        Variable cn
        Variable an
    ▶ Relation Attribute2Column
    ▶ Relation SuperAttribute2Column
```

(b)

```
UML2RDBMS.tal.xmi
▼ platform:/resource/JTLanalyzer/model/UML2RDBMS.tal.xmi
  ▼ Transformation UML2RDBMS
    ▼ Relation Class2Table
      ▼ Domain UML
        ▼ Pattern c
            Predicate is_persistent
            Predicate name
            Predicate attrs
        ▼ Pattern attr
            Predicate name
      ▼ Domain RDBMS
        ▼ Pattern t
            Predicate name
            Predicate col
        ▼ Pattern col
            Predicate name
    ▶ Relation Attribute2Column
    ▶ Relation SuperAttribute2Column
      Model UML
      Model RDBMS
```

Fig. 4: The UML2RDBMS tranformation and analysis models

(1) Translating UML2RDBMS into the analysis language

Starting from the specification of the UML2RDBMS transformation by means of JTL described in Sect. 2 (see Listing 1.1), the corresponding JTL model in Ecore is represented in Fig. 4(a). As aforesaid, in JTL the mapping between candidate models are

[7] The implementation is available at http://jtl.di.univaq.it/

represented by means of a set of relations defined by two domain patterns. Each relation includes a pair of *when* and *where* predicates which specify the pre- and post-conditions that must be satisfied by the elements of the candidate models.

The JTL model in Fig. 4(a) is translated into the correspondent analysis model in Fig. 4(b) by means of an automated model-to-model transformation implemented in ATL. In particular, all the mapping rules (including pre- and post- conditions) of the source JTL model are translated in the corresponding sets of patterns of the target analysis model. For instance, let us consider the *Class2Table* relation, then every predicate belongs to the UML or RDBMS domain pattern, is represented by an OCL expression, and is evaluated as follows: *(i)* each `Object Template Expression` is transformed in a named `Pattern` of the analysis model; *(ii)* each statement is evaluated and transformed in a correspondent predicate; finally *(iii)* each condition is considered as a pattern and added to the correspondent `Domain` in the analysis model.

(2) Executing the analysis of the UML2RDMBMS transformation
The transformation analysis model, generated during the previous step is entered into the ASP-based engine. In order to perform the analysis, model transformation and the involved metamodels have to be encoded in the ASP language. For instance, in Listing 1.3 a fragment of the ASP encoding of the analysis models in Fig. 4(b) is showed. In particular, it declares the `class2table` relation with an identifier `1` (in Lines 1) and the correspondent patterns and predicates for the domain UML (in Lines 2-10) and the patterns and predicates for the domain RDBMS (in Lines 11-17), as well.

```
 1 relation_name(1, class2table).
 2 relation_domain(1, uml).
 3  relation_pattern(1, uml, c, class).
 4   relation_predicate(1, uml, c, is_persistent, true).
 5   relation_predicate(1, uml, c, name, cn).
 6   relation_predicate(1, uml, c, attrs, attr).
 7  relation_pattern(1, uml, attr, attribute).
 8   relation_predicate(1, uml, attr, name, an).
 9   relation_predicate(1, uml, attr, owner, c). % added from where
10   relation_predicate(1, uml, attr, is_primary, false). % added from where
11 relation_domain(1, rdbms).
12  relation_pattern(1, rdbms, t, table).
13   relation_predicate(1, rdbms, t, name, cn).
14   relation_predicate(1, rdbms, t, cols, col).
15  relation_pattern(1, rdbms, col, column).
16   relation_predicate(1, rdbms, col, name, an).
17   relation_predicate(1, rdbms, col, owner, t).
```

Listing 1.3: A fragment of the analysis model encoded in ASP

Starting from the ASP encoding, the analysis is now able to detect the non-deterministic rules. In particular, the analysis outcome consists of a number of rule sets; each of which includes logically related relations whose simultaneous applications may generate multiple solutions. For instance, Listing 1.4 shows a fragment of the result of the analysis executed on the UML2RDBMS specification (in Listing 1.1). In particular, when the transformation is executed in the RDBMS-to-UML direction, the set of non-deterministic relations `class2table`, `superAttribute2column` and `superAttribute2column` is detected. In particular, in the scenario described in Sect. 2, more than one alternative solutions can be obtained because of the simultaneous executions of the three ambiguous mappings.

```
%%%% Analysis Model
%%%% sets of non-deterministic relations (UML to RDBMS direction):
[]
%%%% sets of non-deterministic relations (RDBMS to UML direction):
[(1,class2table), (3,superAttribute2column), (5,superAttribute2column)]
[(..), (...)]
```

Listing 1.4: A fragment of the output of the analysis

The final goal of such analysis is to convey to the transformation implementor enough information to perform a refinement which resolves the non-determinism. In this respect, the ”guilty” rules are opportunely marked in such a way the sets of rules responsible of the non-determinism sources can be easily recognized by the designer.

6 Related work

Non-determinism is a recurrent aspect affecting model transformation design and implementation. As said, most of the current languages (e.g.,[11, 17, 3, 19, 16, 14, 15] are able to generate only one model at time; in this way even if the transformation is non-bijective the solution is normally identified according to details which are often unknown to the designer rendering such approached unpredictable and therefore unpractical. Existing declarative approaches [6, 12] cope with the non-bijectivity by generating sets of models at once, i.e., all the admissible solutions of a transformation are generated as a solution set. Among them, JTL [6] is specifically tailored to support bidirectionality and change propagation. In [12], the authors propose to use Alloy and follows the predictable principle of least change [13], by using a function that calculates the distance between instances reducing the generated models. Non-determinism is also considered in BiFlux [17], which adopts a novel ”bidirectional programming by update” paradigm, where a program succinctly and precisely describes how to update a source document with a target document, such that there is a unique inverse source query for each update program. Within the Triple Graph Grammars, PROGRES [2] considers non-deterministic cases by demanding user intervention to rule execution in order to choose the desired behavior.

Existing approaches and tools involve testing (e.g., test case generation for model transformations), or analysis performed only on executing programs (e.g., run-time monitoring) [1]. In contrast, in [1] the author propose an approach to analyze model transformation specification represented in Alloy. The simulation produces a set of random instances that conform to the well-formedness rules and eventually that satisfy certain properties. Furthermore, existing functional approaches perform model transformation analysis [11, 17], in order to evaluate the transformation validity (generally, in terms of correctness) before its execution.

7 Conclusion and Future Work

In this paper, we have proposed an approach to detect non-determinism in bidirectional transformations at static-time, i.e., without executing the transformation. The intention is to provide transformation implementors with a tool capable of analyzing transformations in order to return feedback which could resolve potential non-determinism. The approach has been demonstrated on a small yet significant case study implemented in JTL. Despite the analysis is completely independent from JTL, in order to make the approach completely language-independent additional implementation efforts are required which we intend to do in the near future. Furthermore, the logical foundation of the engine makes possible the verification of different formal properties by translating them in ASP, with the scope to provide a mean for improving the quality of model transformation specifications.

References

1. K. Anastasakis, B. Bordbar, and J. M. Küster. Analysis of model transformations via Alloy. In *ModeVVa'07*, pages 47–56, 2007.
2. S. M. Becker, S. Herold, S. Lohmann, and B. Westfechtel. A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. *Software and System Modeling*, 6(3):287–315, 2007.
3. A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, and A. Schmitt. Boomerang: Resourceful lenses for string data. In *POPL 2008*, pages 407–419, 2008.
4. G. Callow and R. Kalawsky. A Satisficing Bi-Directional Model Transformation Engine using Mixed Integer Linear Programming. *JOT*, 12(1):1: 1–43, 2013.
5. K. Chen, J. Sztipanovits, S. Abdelwalhed, and E. Jackson. Semantic Anchoring with Model Transformations. In *ECMDA-FA*, 2005.
6. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *SLE10*, pages 183–202, 2010.
7. K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective - GRACE meeting. In *Procs. of ICMT2009*.
8. R. Eramo, A. Pierantonio, and G. Rosa. Uncertainty in bidirectional transformations. In *Procs. of MiSE 2014*, 2014.
9. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Procs of ICLP*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
10. T. Hettel, M. Lawley, and K. Raymond. Model Synchronisation: Definitions for Round-Trip Engineering. In *Procs. of ICMT 2008*, 2008.
11. S. Hidaka, Z. Hu, K. Inaba, H. Kato, and K. Nakano. Groundtram: An integrated framework for developing well-behaved bidirectional model transformations. In *ASE 2011*, 2011.
12. N. Macedo and A. Cunha. Implementing QVT-R Bidirectional Model Transformations Using Alloy. In *FASE*, pages 297–311, 2013.
13. N. Macedo, H. Pacheco, A. Cunha, and J. N. Oliveira. Composing least-change lenses. *ECEASST*, 57, 2013.
14. MediniQVT. http://projects.ikv.de/qvt/.
15. ModelMorf. http://www.tcs-trddc.com/modelmorf/.
16. Object Management Group (OMG). MOF 2.0 QVT Final Adopted Specification v1.1, 2011. OMG Adopted Specification formal/2011-01-01.
17. H. Pacheco and Z. Hu. Biflux: A bidirectional functional update language for XML. In *BIRS workshop: Bi-directional transformations (BX)*, 2013.
18. D. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.
19. A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In *in Proc. of WG94*. Springer, 1995.
20. S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.
21. P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *SOSYM*, 8, 2009.