Improved traceability for bidirectional model transformations ⋆

Romina Eramo, Alfonso Pierantonio, and Michele Tucci

Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
name.surname@univaq.it

Abstract. Conventional wisdom on bidirectionality in Model-Driven Engineering (MDE) suggests that it represents a crucial component to achieve superior model management, whether it be round-tripping, synchronisation, or consistency restoration. Despite their relevance, bidirectional transformations remain difficult to design and implement due to the complexity they must usually encode and their semantic intricacy. Using a proper traceability support enables transformations to be persistent and permits designers to deal with such cases that would be otherwise largely unfeasible. This also implies dealing with the different types of model relationships that may exist in order to establish (re)usable traceability links. This paper proposes to leverage traceability information between source and target elements of a transformation to a first-class status in order to *i)* automatise its generation, *ii)* enable a model-based representation and *iii)* ease reuse and refinement in a further stage. The approach is realized within the JTL framework.

1 Introduction

Conventional wisdom on bidirectional transformations in Model-Driven Engineering (MDE) [13] suggests that they are crucial components to achieve superior model management, whether it be round-tripping, synchronization, or consistency restoration [17, 19]. Rather than writing two unidirectional transformations, which need to be kept consistent, the implementor may provide a single relational specification that can be consistently executed in both directions.

Despite their relevance, bidirectional transformations remain difficult to design and implement due to the complexity they usually encode and their semantic intricacy. In fact, the invertibility of a transformation can be severely affected in case of partiality and/or non-injective mappings [7]. Current tools when dealing with non-injective mappings expose two different kinds of behaviour: either they make arbitrary choice of one consistent model based on heuristics, or they generate all consistent models and let the user pick one. In the latter case, invertibility might be jeopardised if the information about which model is singled out by the user is not made persistent while transforming it in both directions [18]. As to partial transformations, they typically do not cover all the (source) concepts with consequent information loss that may give place to unwanted behaviour when the transformation is reversed. As a matter of fact, practical model transformation engines frequently fail to restore consistency between models [16].

⋆ This research was supported by the ECSEL-JU through the MegaMart2 project (grant agreement No 737494).

Complex round-tripping scenarios are very challenging and tools tend to respond with ad-hoc solutions compromising relevant properties as shown in Sect. 2, where better mechanisms to enforce *persistence* in bidirectional transformations are needed for storing relevant aspects about the transformation execution. This paper proposes a traceability model to maintain persistence in bidirectional transformations. The approach has been realised within the Janus Transformation Language [3] (JTL) framework, that is specifically tailored to support bidirectional model transformations and change propagation even in case of non-determinism [4]. Its relational semantics relies on Answer Set Programming [6] (ASP) that make use of constraint solver to generate one or more solution models starting from a bidirectional transformation specification. In previous works [3, 4], the traceability capabilities of JTL have been introduced as an implicit mechanism of the underlying engine. Such an implicit mechanism has been used to maintain a reference linkage between source and target elements and then discarded after the transformation execution. This paper extends the previous work and presents an advanced traceability approach aiming at: *i)* automatically storing expressive traceability information between source and target elements to prevent transformation *volatility* even in case of partial and non-injective mappings ; *ii)* first-class representation of traceability as separate models, and *iii)* reuse of traceability in the transformation process to enable the invertibility of the transformation and provide a support for managing models as well as their traceability information.

Structure of the paper. Section 2 illustrates a motivating example that is used throughout the paper to demonstrate the approach. Sect. 3 proposes a metamodel for traceability and Sect. 4 describes how our approach has been realized within the JTL framework. In the next sections, related work and some conclusions are presented.

2 Motivating example

Over the last decade, the need for testing and evaluating transformation tools led to the Transformation Tool Contest (TTC) workshop series. During the 2017 edition[1] (held under the umbrella of the STAF Conference[2]) three cases have been selected via single blind reviews in order to assess the expressiveness, the usability, and the performance of transformation tools. In this section, the Families to Persons Case [1] is considered because of the challenges it might pose. The considered bidirectional scenario involves the metamodels Families and Persons (as described in [1]). A family register stores a collection of families whose members are distinguished by their roles, i.e., mother, father, daughters, and sons. Besides that, a person register maintains a flat collection of persons who are either male or female. Families and persons models are kept in a consistent state by means of a bidirectional transformation, so that: i) mothers and daughters (fathers and sons) are mapped to females (males), and vice versa; and ii) the name of every person is composed of the family name and the member name. Furthermore, there may be multiple families with the same name and multiple persons with the same name and birthday.

The metamodels are clearly non-isomorphic since there are not one-to-one correspondences among the concepts in both metamodels; for instance, the birthday attribute in person does not have a counterpart in family members. Moreover, a person may corre-

[1] `http://www.transformation-tool-contest.eu/2017/`

[2] `http://www.informatik.uni-marburg.de/staf2017/`

spond to a parent or a child, and persons may be grouped into families in different ways, which is clearly non-injective. As a consequence, depending on the transformation engine models can be transformed in different ways leading to unwanted behaviours over which designers may have little or no control as shown in the rest of the section.

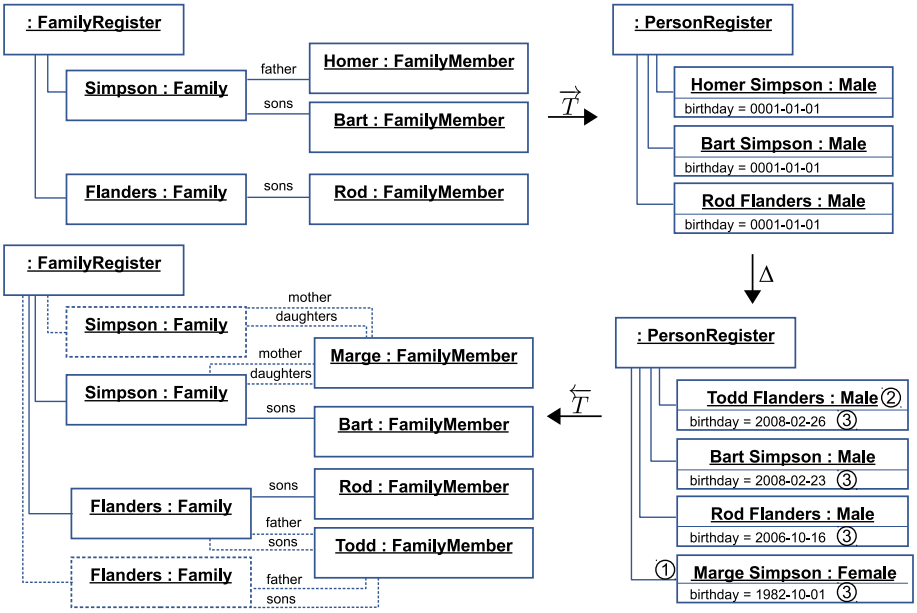

Fig. 1: Sample models for the Families to Persons case

Figure 1 describes a round-trip scenario, where the `Families` model in the upper-left corner consists of the family `Simpson`, with a father `Homer` and a son `Bart`, and a family `Flanders`, with a son `Rod`. The corresponding `Persons` model is given in the upper-right corner: it consists of the corresponding male persons, i.e., `Homer Simpson`, `Bart Simpson` and `Rod Flanders`, whose `birthday`(s) are set to the metamodel default value `0001-01-01` as they do not have a counterpart in the source model. At this point, the `Persons` model is manually modified as follows (see the lower-right part of Fig. 1): ① a new female `Marge Simpson` is added (initially its `birthday` is set to the default value); ② `Homer Simpson` is changed in `Todd Flanders`; ③ the `birthday`(s) of both `Flanders`' and `Simpson`'s are changed. Propagating such changes back to the source model is not univocal as more than one valid update policy is possible as illustrated in the lower-left corner of the same figure where dashed lines denote alternative elements. While, `Bart` and `Rod` are deterministically restored in their original families and role (`sons`), the change ① can be propagated in four different ways: `Marge` can be either `mother` or `daughters` in the existing `Simpson` family or in a new `Simpson` family. Due to the change ②, `Homer` is no longer a member of the `Simpson` family because his full name changed; thus, `Todd` can be equally mapped as a `father` or `sons` either in the existing or the new `Flanders` family. Finally, the change ③ can not be propagated to the source model since `birthday` is missing in the `Family` metamodel.

The non-injective and partial mappings described in the scenario are clearly challenging for most (if not all) existing transformation engines. In order to mitigate the

problem of the uncertainty related to the above scenario, the TTC case authors have provided two configuration parameters controlling the backward transformation: one regulates whether a person is mapped to a parent or a child, while the other whether a person is added to an existing or a new family. However, the problem remains intrinsically difficult to solve and, despite the parameters, the execution of the TTC test cases on several tools highlighted some shortcomings as demonstrated by executing the TTC testsuite on Medini QVT[3] and eMoflon[4]. Both tools make use of some traceability management to enable incremental updates controlled by the configuration parameters. While eMoflon correctly restores already existing elements, Medini is not able to restore the `Bart` and `Rod` original roles and families (see Fig. 1). According to the following configuration parameters settings (`PREFER_CREATING_PARENT_TO_CHILD = true` and `PREFER_EXISTING_FAMILY_TO_NEW = true`) in eMoflon the new female `Marge` is added as `mother` in the existing `Simpson` family; while in Medini `Marge` is wrongly added in a new Family. As to the renaming (cfr. ②), Medini again exposes difficulties in restoring existing roles, whereas eMoflon is not able to configure the preference when both family and role changed. Finally, both approaches are *forgetful* when dealing with the modified birthdays in the target domain (cfr. ③) failing in re-establishing them during a forward execution. While placing reliance on tracing information is well-accepted in order to support round-tripping processes, the mechanisms adopted by these tools seem not accurate and do not help in achieving relevant transformation properties, such as correctness (in the sense of [15]), which is - at the end of the day - the reason because they are adopted. It worths also noting, that the configuration parameters as a mean for controlling transformation behaviour seem problematic because they are subjectively decided and not part of the transformation specification.

Despite the described scenario is not complex, it demonstrates that transformation tools are far from being reliable up to the point that new research directions (e.g., [16]) are exploring sub-optimal characterisations to cope with tool difficulties. In particular, invertibility in bidirectional transformations can be supported by managing traceability information of the transformation executions, which can be exploited later on to trace back the target models. In fact, each generated element can be linked with the corresponding source and contribute to the resolution of some problem. For instance, in Fig. 1 the correspondence between a person and its source role and family is necessary to restore Bart and Rod in their original families as sons. However, no trace links can be maintained for the new elements that cause the generation of multiple alternatives. Finally, information not considered by the partial mapping should be stored so that is not lost in the backward transformation. For instance, the updated birthday dates should be stored in order to be mapped in next alignments. Supporting traceability allows transformations to be persistent and overcome these difficulties.

3 A metamodel for traceability

In this section, we introduce the Trace Metamodel adopted in our approach. It allows to specify links between model elements by using expressive traceability relationships for enabling a correct behaviour in bidirectional transformation executions.

[3] Medini QVT: `http://projects.ikv.de/qvt`
[4] eMoflon: `http://www.moflon.org`

The Trace Metamodel is depicted in Fig. 2; for each transformation execution (both in forward and backward directions) a trace model (`TraceModel`) is generated; it relates a model belonging to a *left domain* to a model belonging to a *right domain*.

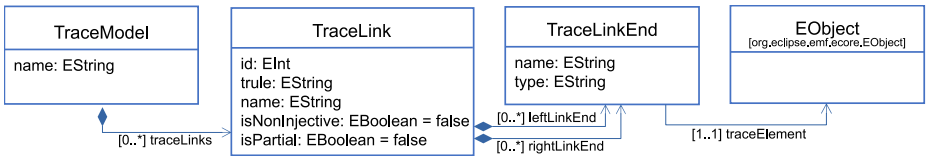

Fig. 2: Trace Metamodel

In particular, a set of trace links between left and right elements and the transformation rule that enforced their mapping is collected. A trace link is characterized by an `id`, a rule name (`trule`), a `name` (i.e., concatenation of `trule` and `id`), a boolean `isNonInjective` (that is true in case of non-injective trace) and a boolean `isPartial` (that is true in case of partial trace). Furthermore, it relates one or more elements belonging to the left domain (`leftLinkEnd`) and the correspondent one or more elements belonging to the right domain (`rightLinkEnd`); such references target elements of type `TraceLinkEnd`, that have a `name` and a `type`. Each `TraceLinkEnd` represents a specific object in the left or right domain (`EObject`).

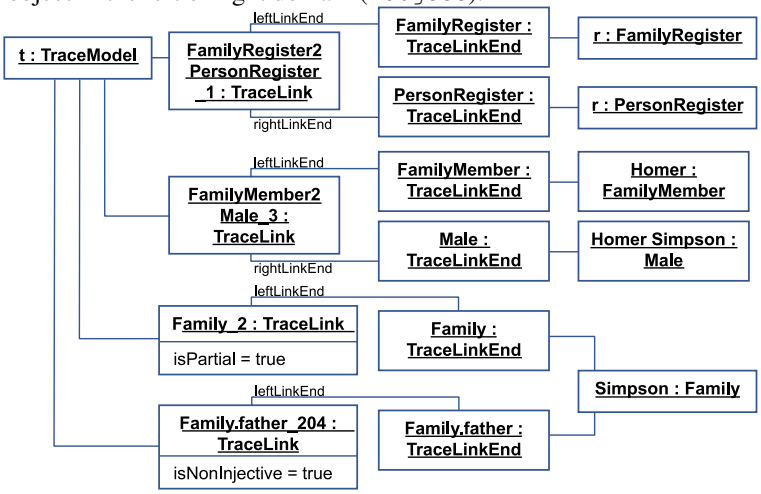

Fig. 3: A sample of trace model

Hereafter, a sample of trace model is depicted (see Fig. 3). It represents a fragment of the trace model generated by the forward execution of the Families to Person case as described in Fig. 1. The model in figure shows the following trace links:

- `FamilyRegister2PersonRegister_1` relates objects `:FamilyRegister` and `:PersonRegister` via the rule named `FamilyRegister2PersonRegister`
- `FamilyMember2Male_3` relates the object `Homer` of type `FamilyMember` to the object `Homer` of type `Male` via the rule named `FamilyMember2Male`
- `Family_2` is a partial link that stores the object `Simpson` of type `Family` that does not have a counterpart in the Persons domain
- `Family.father_204` is a non-injective link that stores the reference `father` of the object `Simpson` of type `Family` that is involved in a non-injective mapping.

At this point, in order to let the JTL engine manage traceability models defined above, we need to extend the framework as follows.

4 Traceability management

In this section, we present our approach to traceability management. It has been realized within JTL (Janus Transformation Language) [3], that is a constraint-based transformation framework specifically tailored to support bidirectionality and change propagation.

4.1 Including traceability in the bidirectional transformation process

The JTL semantic engine is written in the Answer Set Programming (ASP) [6] language and makes use of the the DLV solver [11] to execute transformations in both directions. It is able to find and generate, by a deductive process and in a single execution, all the possible models which are consistent with the transformation rules.

The overall architecture of the JTL environment, upgraded with traceability capability, is reported in Fig. 4. The engine executes JTL transformations which have been written in a QVT-like syntax, and then automatically transformed into ASP programs. Also the source and target metamodels of the considered transformation (MM_1, MM_2) are automatically encoded in ASP and managed by the engine during the execution of the considered transformation. For each transformation execution, a trace model relating source and target model elements is generated and optionally given as input of the next execution (T_{M1M2}, T_{M2M1}). Trace models that conform to the metamodel TTM are automatically mapped in/from ASP. The overall architecture has been implemented as an Eclipse plugin[5] and mainly exploits the Eclipse Modelling Framework (EMF) [2].

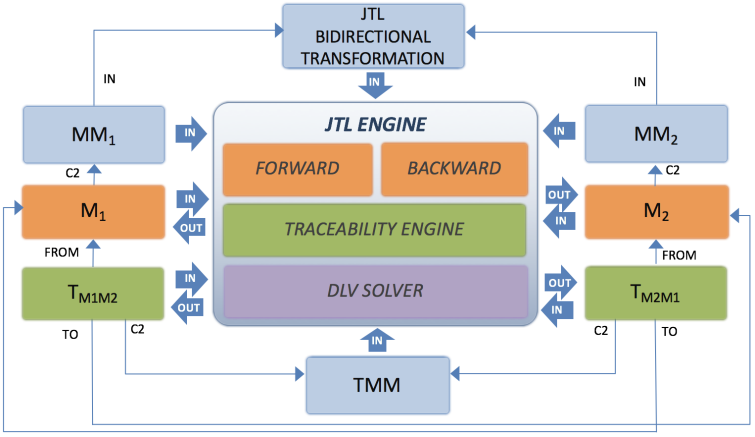

Fig. 4: Overview of the JTL framework with traceability

The traceability management is composed of the following phases:

- *Automatic generation:* The traceability mechanism exploits the intrinsic characteristic of the ASP-based engine to derive relevant details about the linkage between source and target model elements at execution-time (including the applied transformation rules). This step relies on the $Traceability\ Engine$;
- *Explicit representation and storing:* Trace links are extrapolated during the transformation execution and made explicit by the framework. In particular, the ASP

[5] JTL: http://jtl.di.univaq.it

trace links are automatically mapped in Ecore. That is, trace models are maintained as models conforms to the Trace Metamodel, as defined within EMF (see Sect. 3);
- *Re-use:* Trace models can be re-used during the transformation execution. In particular, a trace model can be given as input of the transformation execution in order to (re-)establish consistency, manage ambiguities and guarantee the correctness of the transformation. In particular, the $Traceability\ Engine$ is able to include traceability information in the execution process.

The traceability management is shown in practice in the next section. In particular, the proposed approach to the *Families to Persons* case was presented in Sect. 2 with the intent to illustrate how it works in round-trip scenarios.

4.2 Families to Persons round-tripping

According to the scenario described in Sect. 2, the input model conforming to the *Families* metamodel is reported in its Ecore representation in the left-hand side of Fig. 5. The application of the JTL *Families2Persons* bidirectional transformation [6]. on `sampleFamily` generates the corresponding `samplePerson` model as depicted in the right part of Fig. 5. At the same time, the trace model `sampleFamily2samplePerson`, depicted in the middle of the Fig. 5, is generated (dashed arrows connect trace elements to the elements they refer to in source and target models).

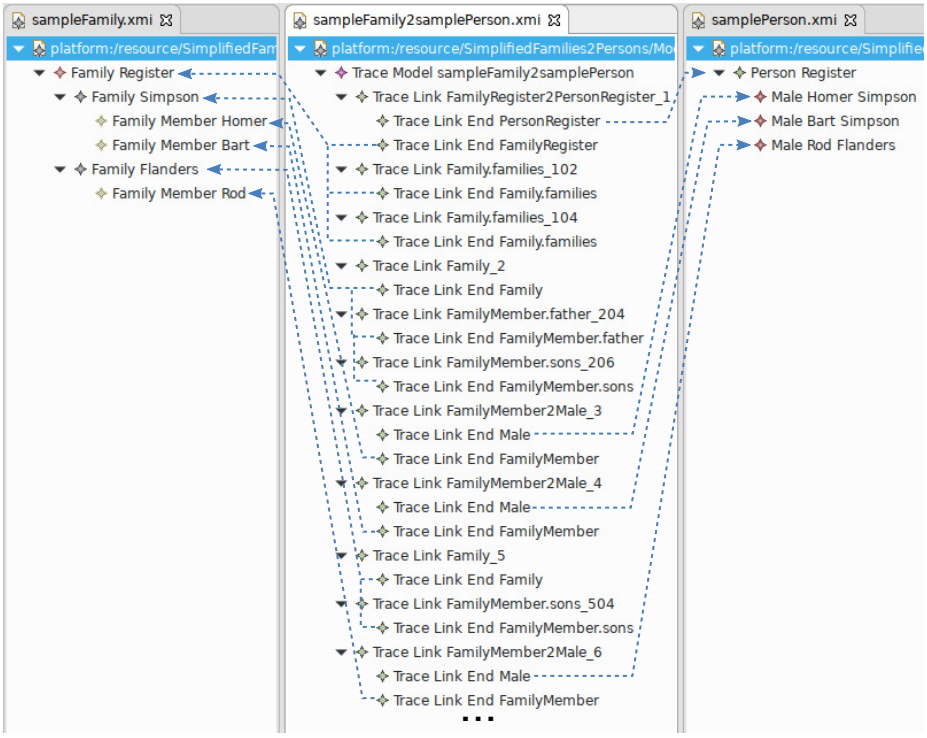

Fig. 5: The forward execution of the Families to Persons case

The generated trace model is composed of: *a)* trace links `FamilyRegister2PersonRegister_1`, `FamilyMember2Male_3`, `FamilyMember2Male_4`, and `FamilyMember2Male_6`, that store how family elements (left `TraceLinkEnds`) have been

[6] The interested reader can access the implementation at http://jtl.di.univaq.it/

mapped to person elements (right `TraceLinkEnds`), *b)* partial trace links `Family_families_102`, `Family_families_104`, `Family_2`, `Family_5`, that represent elements or structural features not covered by the transformation and *c)* non-injective trace links `FamilyMember_father_204`, `FamilyMember_sons_206`, `FamilyMember_sons_504` store elements involved in non-injective mappings. Note that, by re-applying the transformation in the backward direction it is possible to obtain again the `sampleFamily` source model. The missing information about the original families and role of the members are restored by means of trace information, that is giving as input the trace model `sampleFamily2samplePerson` obtained in the previous forward execution.

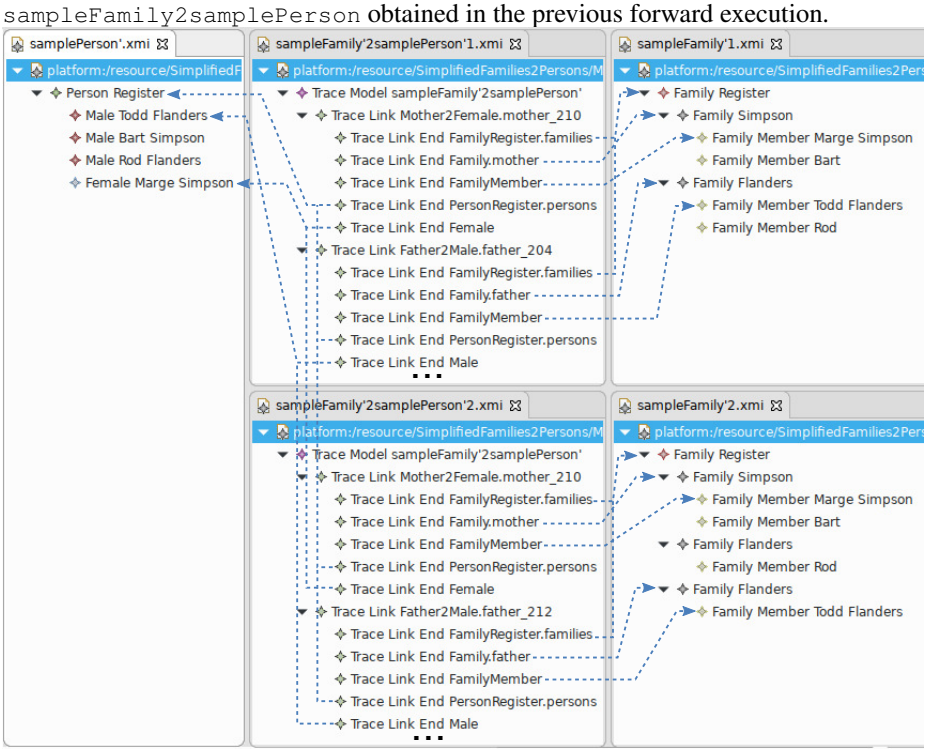

Fig. 6: The backward execution of the Families to Persons case

After a refinement step, the target model including the changes described in Sect. 2 is shown in the left part of Fig. 6. The modified model `samplePerson'` and the trace model `sampleFamily2samplePerson` (obtained in the previous execution) are given as input of the backward transformation execution. Due to the non-injectivity, 16 alternative models, namely `sampleFamily'{1,2,..,16}` are generated. All the alternative models correctly restore families and roles of the members as in the original source model. Furthermore, the new female Marge Simpson can be propagated both as a mother or a daughter in the existing Simpson family as well as in a new one. Analogously, the changed member `Todd Flanders` can be propagated in four different ways by adding Todd to the existing or to a new family either as father or as a son. Finally, for each generated model, a correspondent trace model is generated. In the middle and right side of the Fig. 6, two representative generated models and a fragment of their traces are illustrated. In both family models, Marge Simpson is created as mother of the existing

Simpson family, as stored in the trace link `Mother2Female.mother_210` in both the trace models. Moreover, in the model `sampleFamily'1` Todd Flanders is created as father of the existing Flanders family, while in the model `sampleFamily'2` he is created as father of a new family, as stored in the trace links `Father2Male.father_204` and `Father2Male.father_212` in the first and second trace models, respectively. Note that, by selecting one of the alternative models and re-applying the transformation in the forward direction it is possible to obtain again the `samplePerson'` model. The missing information about the birthday attributes are restored by means of trace information, that is giving as input the trace model `sampleFamily'12samplePerson'` obtained in the previous forward execution.

5 Related work

The technological importance of traceability is well recognized in model transformations. Most existing transformation approaches adopt an internal traceability model that can be interrogated at execution time; for instance, to resolve dependencies between the rules or to check if a target elements was already created for a given source element. Widely used unidirectional languages as ATL [9] and ETL [10] automatically create traceability links during the transformation execution. The traceability approach is specific to each transformation language; both in ATL and ETL, relations have to be bijective and traceability is used to resolve dependencies between the rules.

In bidirectional transformations, the state may be stored in explicit or implicit way. Medini QVT uses a trace model to facilitate model transformations and conformance checking, and to enable also incremental updates during further executions. The traceability metamodel is deduced from the transformation specification. In QVT-R relations may not be bijective, however during the execution it only addresses bijective relations. Thus, trace data are not used to the resolution of some ambiguities. In TGG [14], the correspondence graph ensures traceability between source and target elements and traceability can be used to check the correspondence between two models and to synchronize models incrementally. eMoflon provides a Java-based API via which the designer can choose the priority matches as soon as there are multiple choices available in the transformation process. Tracing information are not used to support partial and non-injective cases. Traceability is an intrinsic property in lens-based approach [5, 8, 12]; in fact each element of the abstract structure is related with a correspondent element in the concrete structure. In particular, the put function updates the concrete structures restoring any information discarded during the forward transformation. When a change is performed, the transformation engine propagates the change towards the other direction. In Boomerang [7] and BiFlux [8], traceability information is not explicitly visualizable or stored in any files. In GRoundTram [9], trace information can be generated during the transformation execution as a graph (or textual) file. However, in such functional approaches no support to partial and non-injective transformations is given.

6 Conclusion and future work

Bidirectional transformations are a key mechanism to maintain consistency among interrelated models. However, they remain difficult to design and implement due to the

[7] Boomerang: http://www.seas.upenn.edu/ harmony/
[8] BiFlux: http://www.prg.nii.ac.jp/projects/BiFluX/
[9] GRoundTram: http://www.biglab.org/

intrinsic complexity they must encode. This paper proposed to employ traceability information in order to enable persistent bidirectional transformations. The proposed approach extended the JTL framework by: *i)* automatize the generation of expressive traceability information, *ii)* giving and explicit model-based representation and *iii)* ease reuse and refinement in further stage to enforce consistency restoration. As future work, we plan to extend the approach to work when the traceability model describes different types of traceability links. Furthermore, we are interested in describing constraints over traceability links and maintain that through model transformations.

References

1. Anthony Anjorin, Thomas Buchmann, and Bernhard Westfechtel. The families to persons case. In *Procs. of the 10th Transformation Tool Contest (TTC 2017)*, pages 27–34, 2017.
2. F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T.J. Grose. *Eclipse Modeling Framework*. Addison Wesley, 2003.
3. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *SLE10*, pages 183–202, 2010.
4. R. Eramo, A. Pierantonio, and G. Rosa. Managing uncertainty in bidirectional model transformations. In *Procs. of SLE 2015*, pages 49–58, 2015.
5. J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3), 2007.
6. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Procs of ICLP*, pages 1070–1080, 1988.
7. T. Hettel, M. Lawley, and K. Raymond. Model Synchronisation: Definitions for Round-Trip Engineering. In *Procs. of ICMT 2008*, 2008.
8. 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.
9. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: a Model Transformation Tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.
10. Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. The epsilon transformation language. In *Theory and Practice of Model Transformations*, pages 46–60, 2008.
11. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. 2004.
12. Hugo Pacheco and Zhenjiang Hu. Biflux: A bidirectional functional update language for XML. In *BIRS workshop: Bi-directional transformations (BX)*, 2013.
13. D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.
14. Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1995.
15. Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *SOSYM*, 8, 2009.
16. Perdita Stevens. Bidirectionally Tolerating Inconsistency - Partial Transformations. *FASE*, 8411(Chapter 3):32–46, 2014.
17. Perdita Stevens. Bidirectional transformations in the large. In *Int. Conf. on Model Driven Engineering Languages and Systems, MODELS 2017, 2017*, pages 1–11, 2017.
18. Bernhard Westfechtel. A case study for evaluating bidirectional transformations in QVT relations. In *ENASE 2015*, pages 141–155, 2015.
19. Yingfei Xiong, Hui Song, Zhenjiang Hu, and Masato Takeichi. Supporting parallel updates with bidirectional model transformations. In *ICMT 2009*, 2009.