Generating model with uncertainty by means of JTL

Gianni Rosa
Università degli Studi dell'Aquila
I-67100 L'Aquila, Italy
Email:gianni.rosa@univaq.it

***Abstract*—In Model-Driven Engineering, the potential advantages of using bidirectional transformations are largely recognized. Despite its crucial function, in certain cases bidirectionality has somewhat limited success because of the ambivalence concerning non-bijectivity. In fact, consistently propagating changes from one side to the other is typically non univocal as more than one correct solution is admitted. This gives place to a form of *uncertainty* which means that, rather than having a single model, we actually have a set of possible models but we do not know what is the right one. In this paper, we discuss how dealing with multiple solutions is important and requires specialized tools and support. In particular, handling a set of models explicitly is generally non-viable. Thus, we extended the JTL semantics to generate a model with uncertainty which is semantically equivalent to the set of models it represents. The approach is implemented and a metamodel-independent technique is proposed.**

I. Problem and Motivation

In Model-Driven Engineering [20] (MDE) bidirectionality in transformations has been always regarded as a key mechanism [21]. 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. Probably the main reason why bidirectional techniques had limited success can be found to some extent in the ambivalence concerning non-bijectivity. For instance, while MDE requirements demand enough expressiveness to write non-bijective transformations [23], the QVT standard is somewhat uncertain in asserting whether the language permits such transformations [22]. In particular, when reversing a non-injective bidirectional mapping more than one admissible solution can be found. This gives place to a form of *uncertainty* (as known in [18]): rather than having a single model, we actually have a set of possible models and we are not sure which is the *right* one. On the other hand, while a transformation can always be disambiguated at design-time by fixing those details that leave the solution open to multiple alternatives, in many cases this is non-viable because the designer does not detain enough information beforehand for establishing a general solution. Thus, harnessing declarative approaches capable of dealing with the intrinsic uncertainty of non-bijective bidirectional transformations in a rigorous and precise way can be key to success. Recently, few declarative approaches [2], [3], [16] 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 [3] (JTL) is a model transformation language specifically tailored to support bidirectionality and change propagation. The problem of managing a set of models explicitly is impractical as its size might be quite large, designers need to be supported with suitable mechanism and tools in order to avoid the effect of having multiple design alternatives.

The solution proposed in this paper, is an extension of the JTL semantics and of its engine capable of generating a uniform characterization of the solution in terms of models with uncertainty instead of a set of models, such that *a)* the approach is metamodel-independent, in sense that starting from an arbitrary base metamodel can automatically generate the corresponding uncertainty metamodel; *b)* the resulting model conforms to the uncertainty metamodel; and *c)* each model with uncertainty has to be considered semantically equivalent to the set of its *concretizations*, i.e. the models obtainable from the uncertainty model by resolving the point of uncertainties.

By way of example, consider the *Collapse/Expand State Diagrams* round-trip benchmark [4]. The forward transformation $\overrightarrow{T}$ translates the hierarchical state machine in Fig. 1(a) into the flatten version in Fig. 1(b). $\overrightarrow{T}$ is clearly non-injective because different hierarchical machines can be translated into the same model. Let us suppose now that the designer wants to manually modify the target model by means of the changes Δ highlighted in bold and with thicker lines in Fig. 1(c). More in details, Δ consists of the following modifications:

- a new state `Printing` is added,
- a new transition `print` from `Active` to `Printing` is added,
- the transition `done` from `Active` to `Idle` is deleted and replaced by a new transition `completed`,
- a new transition `done` from `Printing` to `Idle` is added, and finally
- a new transition `critical_error` from `Out of services` to the initial state `Off` is added.

At this point, the original source model and the revised target model are not consistent any longer. Therefore, the backward transformation $\overleftarrow{T}^{-1}$ can be used to restore the consistency by propagating the changes in Δ. Not surprisingly, there is

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

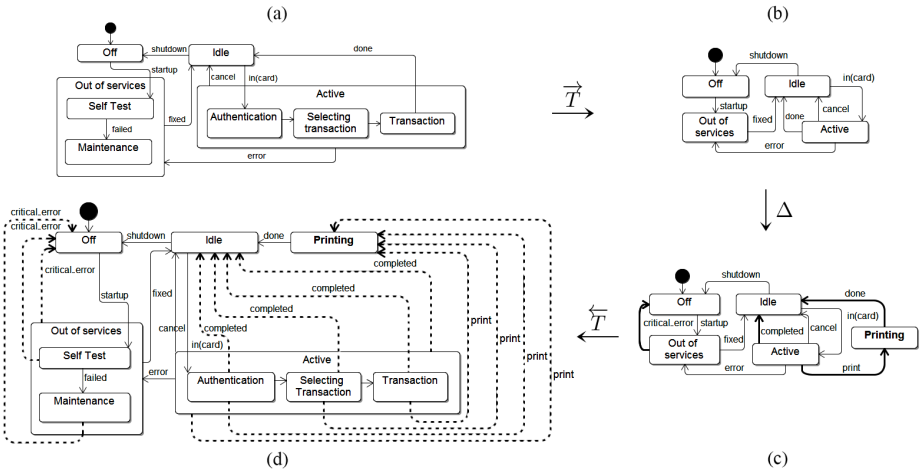

Fig. 1. Collapse/expand state diagrams in a round-trip process

not a unique way of updating the source model. In fact, the added transitions in the target may be mapped to either of the nested states as well as to the container state itself, as illustrated in Fig. 1(d), where the dotted edges represent the alternative transitions. Despite the changes on the target model are relatively simple, their impact on the source model is typically exponential. In fact, the overall number of admissible models in this case is

$$|\texttt{print}| \times |\texttt{completed}| \times |\texttt{critical\_error}| = 4 \times 4 \times 3 = 48$$

where $|\texttt{name}|$ is the number of alternative model elements called `name`. It is worth noting that the models in Fig. 1(d) are represented by means of the dotted notation, which is informal: in this case a bidirectional transformation implemented in JTL would generate a collection of 48 distinguished models. Clearly, whenever the implementor is unable to disambiguate the transformation by making it deterministic, the decision must be left to the modeler rather than to the language internals.

It is not difficult to demonstrate that for a specific instance, if n is the number of uncertainty points and m the number of alternative model elements for each of them, then the result consists of m^n different alternative solutions. Hence, it is of crucial relevance that modelers are adequately assisted in dealing with the *combinatorial explosion* of design alternatives as described in the sequel.

The paper is organized as follows. Section II describes related work about this topic, Section III presents the proposed approach to represent uncertainty by means of JTL, Section IV proposes a new semantic for the JTL engine able to generate and manage models with uncertainty. Finally, Section V draws some conclusion and future work.

II. Related work

Uncertainty is ubiquitous within contexts such as requirements engineering [6], software processes [15] and adaptive systems [19]. Uncertainty management has been studied in many works, often with the intention to express and represent it in models. In [10], the notion of *partial model* is introduced in order to let the designer specify uncertain information by means of a base model enriched with annotations and first-order logic. Model transformation techniques typically operate under the assumption that models do not contain uncertainty. Nevertheless, the work in [11] proposes a technique for adapting existing model transformations in order to deal with models containing uncertainty. The main is a lifting operation, which permits to adapt unidirectional transformations for being used over models with uncertainty preserving their original behavior. In [17] a formal approach called MAVO is proposed and applied to design models in order to express and allow automated reasoning in presence of uncertainty.

Concerning the multiplicity of solutions in bidirectional transformations, most of the existing languages are deterministic, i.e., they produce one model at a time. However, in [1] the authors propose PROGRES, a bidirectional transformation solution based on Triple Graph Grammars (TGGs) which is able to recognize ambiguous mappings and in case resolve them by interactively asking the user, who need to be an expert in these techniques. In [22] the QVT-R bidirectional transformation language is discussed. In particular, the author observes that the formal semantics of QVT-R is ambiguous and it is not possibile to conclude that QVT-R supports non-bijective transformations. An attempt in making bidirectional transformation deterministic by means of intentional updates is represented by the BiFluX language [24], however the problem that a transformation cannot be tested for non-determinism at static-time reduces its effectiveness. Recently some interesting solutions based on *lenses* have been proposed: [5] illustrates a technique to support bidirectional transformations relying no more on mapping between models but across manipulations (or differences) operable on them. However, the management of non-bijective problems is not clearly addressed. Although researchers are actively working on bidirectional transformations in several communities [14], a lot effort must be made to make such kind of transformations ripe for the industrial en-

vironment [13]. Finally, the ability to deduce and generate all the possible solutions of an uncertain transformation has been achieved by few approaches, including JTL [7], [8]. In [16] the authors propose a bidirectional transformation approach in which the QVT-R semantics is implemented by means of Alloy. Different generated alternatives may be obtained from the execution of a model transformation and reduced by adding extra OCL constraints or by limiting the upper-bound search criteria. While in [2] similar results are obtained by using a variety of integer linear programming. These approaches introduced over the last few years on one hand demonstrate that there is a need for an in-depth discussion about the nature of bidirectionality; and on the other hand, shown how mature techniques and semantics are available for dealing with an intrinsically difficult problem.

III. Approach and Uniqueness

The main challenge to address in order to let JTL generate the model with uncertainty is to detect those model elements which are shared throughout the solution space, i.e., the certain part of the solution. Referring to Fig. 1(d), it consists of all elements in the model but the dotted edges. This is obtained by logically connecting those elements in the source model which originated the same target elements. Since the JTL engine is a logic program written in ASP, it can derive how models are related by means of a deductive process. In this respect, the existing traceability management offers enough information to understand how the models can be factorized by identifying those elements which are connected among them. In particular, tracing information stores relevant details about the linkage between source and target model elements at execution-time (including the applied transformation rules).

An excerpt of the implemented mechanism is given in Listing 1. In particular, it contains (part) of the ASP encoding of the transitions *print* present in the modified simple machine in Fig. 1.(c) (line 1) and in the hierarchical state machine in Fig. 1.(d) (lines 3-6); the corresponding trace links are given in lines 8-11. During the execution of the transformation, the engine is able to deduce both the concretization models and/or the model with uncertainty. Any point of uncertainty is derived by calculating the uncertainty set among model elements (lines 15-24).

```
1  node(SM, p, transition).
2
3  node(HSM, p1, transition).
4  node(HSM, p2, transition).
5  node(HSM, p3, transition).
6  node(HSM, p4, transition).
7
8  trace_link(tl1,s1,t1,r1).
9  trace_link(tl2,s1,t2,r2).
10 trace_link(tl1,s1,t3,r3).
11 trace_link(tl2,s1,t4,r4).
12
13 getID(p,s1). getID(p1,t1). getID(p2,t2). getID(p3,t3).
       getID(p4,t4).
14
15 get_uncertainty_set(t1,t2) :-
16 trace_link(tl1,s1,t1), trace_link(tl2,s2,t2),
17 equals_id(s1,s2), equals_id(t1,t2).
18
19 node(UMM,ID,UMC) :-
20 get_uncertainty_set(t1,t2), getID(ID,t1), getUMM(UMM,t1),
       getUMC(UMC,t1).
21
22 edge(UMM,IDe,URef,ID,IDt) :-
23 get_uncertainty_set(t1,t2), getID(ID,t1), getUMM(UMM,t1),
       getRefID(IDe,ID),
24 getChildID(IDt,ID), getURef(URef,t1).
```

Listing 1. A fragment of the adapted JTL engine

With reference to the scenario described in Sect. I, the model with uncertainty corresponding to the admissible 48 solutions, is shown in Fig. 2. In particular, the alternative transitions are collected in the uncertainty point (`UTransition`) `print`, `completed`, `new operation` which contains the transitions targeting each one of the nested states within `Active` as well as to the composite state itself.

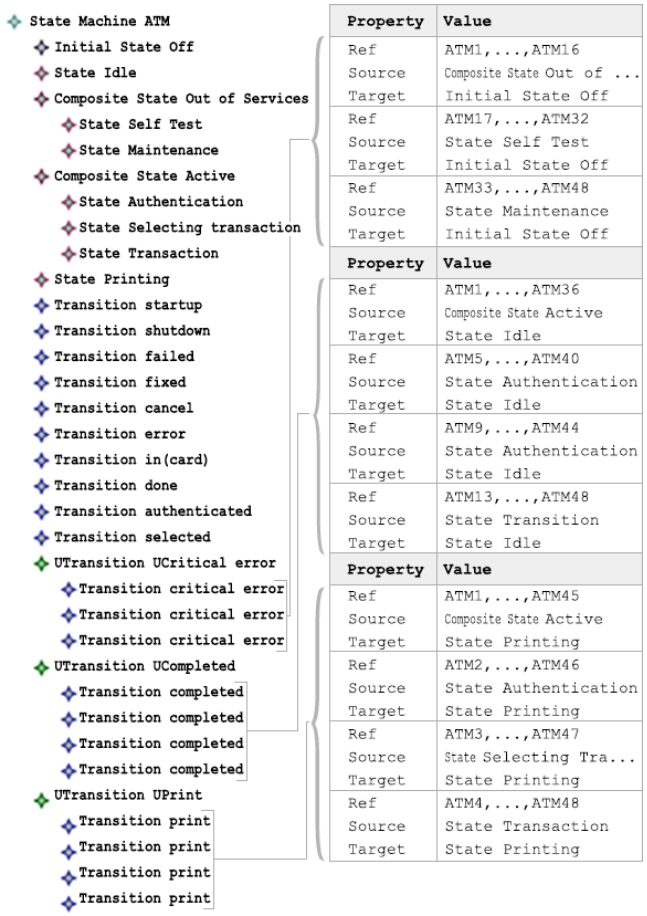

Fig. 2. UHSMm model

Due to the metamodel structure (Fig.4), models with uncertainty may over-approximate the sets of transformation candidates. For instance, the scenario in Fig. 1 suggests that only one `print` transition can exist in the final model. However, the generated model with uncertainty admits also models with multiple `print` transitions giving place to more concretizations than those expected. Therefore, in order to avoid multiple `print` transitions, an operation of refining with a constraint on the model is needed in order to reduce the concretizations to cases with one `print` transition only. The

constraints are directly generated by the JTL engine, which possesses all the information in the trace links between models source and target.

IV. Results

It is worth noting how uncertainty is becoming increasingly important in today's software based systems. Rather than ignoring uncertainty, it should be considered as a first-class concern in the design, implementation, and deployment of those systems [12]. Typically, it occurs when the designer does not have complete, consistent and accurate information required to take a decision during any stage of software development. Introducing uncertainty in modeling processes means that, rather than having a single model, designer actually have a set of possible models and she is not sure which is the correct one [10]. In bidirectional transformations, uncertainty becomes manifest only after the transformation is executed but clearly originates from the inability of providing intentional updates at design-time. The idea of representing a set of solutions with a single model with uncertainty is the starting point for extending a language like JTL.

In order to reduce the burden of managing a collection of models, a metamodel-independent approach to uncertainty representation based on preliminary work in [9] is shown below.

The uncertainty metamodel $U(M)$ is obtained by extending a *base* metamodel M with specific connectives to represent the multiple outcomes of a transformation. These connectives denote the uncertainty points where alternative model elements are attached. Moreover, such points of uncertainty are traceable in order to ease the traversal of the solution space and to permit the identification of specific concretizations, i.e., instances. Let us consider the metamodel *HSM* of the hierarchical state machines given in Fig. 3. The uncertainty metamodel $U(HSM)$ in Fig. 4 is automatically obtained by extending the base metamodel as follows:

1) the abstract metaclass `TracedClass` with attributes `trace` and `ref` is added,
2) for each metaclass *c* in *HSM*, such that it is non-abstract and does not specialize other metaclasses:
   2.1) a direct sub-metaclass *uc* of *c* is added,
   2.2) *c* is generalized by `TracedClass`,
3) each metaclass *uc* is composed with *c*, enabling the representation of a point of uncertainty and its alternatives, finally
4) the cardinality of attributes and references are relaxed and made optional in order to permit to express uncertainty also over them.

In particular, the metaclasses `UStateMachine`, `UState` and `UTransition` in $U(HSM)$ are derived from `StateMachine`, `State` and `Transition` in *HSM*, which are in turn generalized by `TracedClass`. The purpose of `TracedClass` is to maintain information about the relationships between the uncertainty points and the correspondent own alternatives in the concretization models.

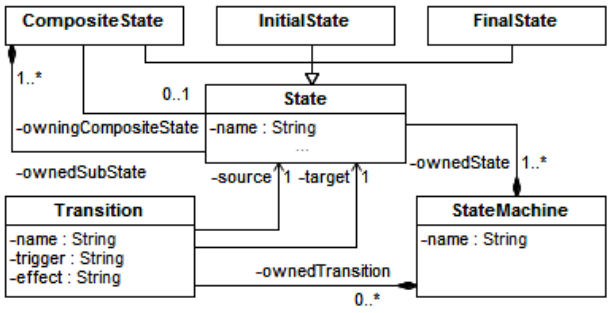

Fig. 3. The *HSM* metamodel

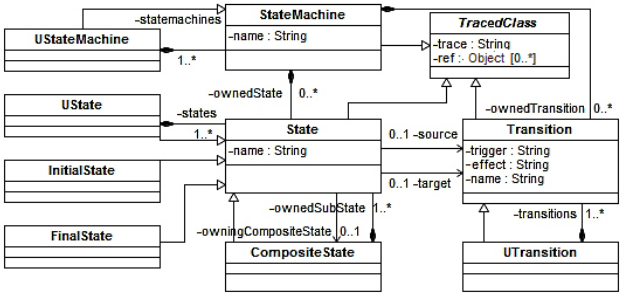

Fig. 4. The $U(HSM)$ metamodel

The above metamodel permits to represent with a single model a set of state machines. For instance, in Fig. 1(d) a set of hierarchical machines are given. Most of the model elements (all the nodes and part of the transitions) are shared among them, they represent the certain part of the models. Whilst, the dotted alternative transitions denoted by `print`, `completed`, and `critical_error` correspond to three different uncertainty points of type `UTransition`, each connected to the related alternative `Transition` elements.

The main advantage of the approach is that the uncertainty represented by a set of alternative models is leveraged to a first-class status. A complete set of models can therefore be manipulated as whole, for instance with an automated transformation, without iterating over each individual in the set. The above discussion captures the fact that non-injective mappings when reversed can produce multiple results. However, as discussed in the previous section managing a multitude of models is impractical. What is needed is a construction capable of representing a set of models generated by a transformation in an intensional way. Hence, the revised JTL engine is able to distinguish among two different behaviors:

- *extensional*, generate all the models satisfying the relation defined in the bidirectional transformation;
- *intensional* (or with *uncertainty*), generate a model with uncertainty which is semantically equivalent to the models of the extensional case, i.e., the corresponding set of models can be generated from it.

However, one important issue arising from this scenario is that models with uncertainty may be over-approximations

of the sets of transformation candidates. This is due to the "combinatorial" nature of these models since each point of uncertainty collects the different alternatives. Consequently, it can happen that certain combinations produce concretizations which are not part of the extensional solution space. Therefore, besides the models with uncertainty it is important to generate also those constraints which limit the solution to the admissible concretization only.

V. Conclusion

Bidirectional model transformations represent at the same time an intrinsically difficult problem and a crucial mechanism for keeping consistent and synchronized a number of related models. One of the prevalent factors within software engineering is the problem of non-determinism in bidirectional transformations. When modellers are not able to fix a design decision they may encode ambiguities in their model transformation specification, e.g. not providing additional constraints that would make the transformation deterministic. The lack of information affects models, for instance ambiguous mapping may cause the generation of multiple solution models each one representing a different design decision.

Therefore, the proposed approach is based on the refinement of the JTL engine, which permits the generation of models with uncertainty in order to leverage the solution space to a first-class concern. The approach has been implemented and can be used for any Ecore artifact.

Acknowledgment

This research was supported by the EU through the Model-Based Social Learning for Public Administrations (Learn Pad) FP7 STREP project (619583)[2].

References

[1] 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.

[2] G. Callow and R. Kalawsky. A Satisficing Bi-Directional Model Transformation Engine using Mixed Integer Linear Programming. *Journal of Object Technology*, 12(1):1: 1–43, 2013.

[3] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *Procs. of SLE 2010*, LNCS 6563, pages 183–202. Springer, 2011.

[4] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective - GRACE meeting notes, state of the art, and outlook. In *Procs. of ICMT2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.

[5] Z. Diskin, Y. Xiong, and K. Czarnecki. From state- to delta-based bidirectional model transformations. pages 61–76, 2010.

[6] C. Ebert and J. D. Man. Requirements uncertainty: influencing factors and concrete improvements. In *Procs. of ICSE*, pages 553–560. ACM Press, 2005.

[7] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. A model-driven approach to automate the propagation of changes among Architecture Description Languages. *SOSYM*, 1(25):1619–1366, 2010.

[8] R. Eramo, A. Pierantonio, J. R. Romero, and A. Vallecillo. Change management in multi-viewpoint system using asp. In *Procs of EDOCW*, pages 433–440, Washington, DC, USA, 2008. IEEE Computer Society.

[9] R. Eramo, A. Pierantonio, and G. Rosa. Uncertainty in bidirectional transformations. In *Procs. of MiSE 2014*, 2014.

[2] www.learnpad.eu

[10] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *ICSE*, pages 573–583, 2012.

[11] M. Famelis, R. Salay, A. D. Sandro, and M. Chechik. Transformation of models containing uncertainty. In *MoDELS*, pages 673–689, 2013.

[12] D. Garlan. Software engineering in an uncertain world. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 125–128. ACM, 2010.

[13] J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 633–642. IEEE, 2011.

[14] D. Hutchison, T. Kanade, J. Kittler, J. Kleinberg, F. Mattern, J. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, et al. Bidirectional transformations: A cross-discipline perspective, grace meeting notes, state of the art, and outlook. In *Theory and Practice of Model Transformations, Second International Conference, ICMT 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings*, volume 5563. Springer, 2009.

[15] H. Ibrahim, B. H. Far, A. Eberlein, and Y. Daradkeh. Uncertainty management in software engineering: Past, present, and future. In *CCECE*, pages 7–12. IEEE, 2009.

[16] N. Macedo and A. Cunha. Implementing qvt-r bidirectional model transformations using alloy. In V. Cortellessa and D. Varró, editors, *FASE*, volume 7793 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2013.

[17] R. Salay, M. Chechik, J. Horkoff, and A. D. Sandro. Managing requirements uncertainty with partial models. *Requir. Eng.*, 18(2):107–128, 2013.

[18] R. Salay, M. Famelis, and M. Chechik. Language independent refinement using partial modeling. In *FASE*, pages 224–239, 2012.

[19] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems. In *RE*, pages 95–103. IEEE Computer Society, 2010.

[20] D. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[21] S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.

[22] P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and Systems Modeling*, 8, 2009.

[23] S. Witkop. MDA users' requirement for QVT transformations. In *OMG doc 05-02-04*, 2005.

[24] T. Zan, H. Pacheco, and Z. Hu. Writing bidirectional model transformations as intentional updates. In *ICSE Companion*, pages 488–491, 2014.