# Representing Uncertainty in Bidirectional Transformations

Romina Eramo, Alfonso Pierantonio, and Gianni Rosa

DISIM University of L'Aquila, Via Vetoio, 67100, Italy,
name.surname@univaq.it

**Abstract.** In Model-Driven Engineering, the potential advantages of using bidirectional transformations are largely recognized. The non-deterministic nature of bidirectionality represents a key aspect: i.e, consistently propagating changes from one side to the other is typically non univocal and more than one correct solutions are admitted. In this paper, the problem of uncertainty in bidirectional transformations is discussed. In particular, we illustrate how represent a family of cohesive models, generated as output of a bidirectional transformation, by means of models with uncertainty.

## 1 Introduction

In Model-Driven Engineering (MDE) [20], the potential advantages of using bidirectional transformations in various scenarios are largely recognized. As for instance, assuring the overall consistency of a set of interrelated models which requires the capability of propagating changes *back* and *forth* the transformation chain [21]. Despite its relevance, bidirectionality has rarely produced anticipated benefits as demonstrated by the lack of a leading language comparable, for instance, to ATL for unidirectional transformations due to the ambivalence concerning non-bijectivity that give place to *non-determinism*. For instance, while MDE requirements demand enough expressiveness to write non-bijective transformations [24], the QVT standard is somewhat uncertain in asserting whether the language permits such transformations [23]. In particular, when reversing a non-injective bidirectional mapping more than one admissible solution might be found. Thus, rather than having a single model, we actually have a set of possible models and we are not sure which is the desired 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 impractical because the designer does not detain enough information beforehand for establishing a general solution. Recently, few declarative approaches [4, 17, 3] 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 [4] (JTL) is a model transformation language specifically tailored to support bidirectionality and change propagation. Unfortunately, managing a set of models explicitly is challenging and poses severe issues as its size might be quite large. The main problem of these approaches is related to the difficulty to manage a number of models.

A similar problem is present in the management of *uncertainty* [12]. Typically it occurs when the designer has not complete, consistent and accurate information required to make a decision during software development. In particular, the designer may add ambiguity during the writing of model transformation. For instance, she may be unsure

about some correspondences among source and target elements and, as a consequence, the transformation may generate multiple solutions each one representing a different design decision. Furthermore, often she can understand it only at execution time, when a multitude of models are obtained.

Providing a representation of the uncertainty generated as outcome of a model transformation process represents a first step to support designers. We propose a metamodel-independent approach to represent a family of cohesive models deriving from a bidirectional transformation by means of models with uncertainty. This paper is related to previous work [11] which introduced the uncertainty due to non-deterministic bidirectional transformations and outlined the challenges aiming to give a support to the problem.

The paper is organized as follows. Section 2 introduces the problem by means of a running example based on JTL. Section 3 discusses uncertainty and the need of a tool support for manage it. Section 4 presents the proposed approach to represent uncertainty. Finally, Sect. 5 describes related work and Sect. 6 draws some conclusion and future work.

## 2 A motivating example

It is more the rule than the exception that uncertainty is part of almost any aspects of software development [22]. This is also valid for model transformation design and implementation. In particular, in this section we describe how lack of information at design-time can lead to non-deterministic transformations which generates uncertainty in the solution because some mapping between elements in models may be ambiguous.

To better understand the problem and the difficulties it poses, we firstly introduce a well-known application scenario and secondly provide an implementation with JTL highlighting its intrinsic non-determinism.

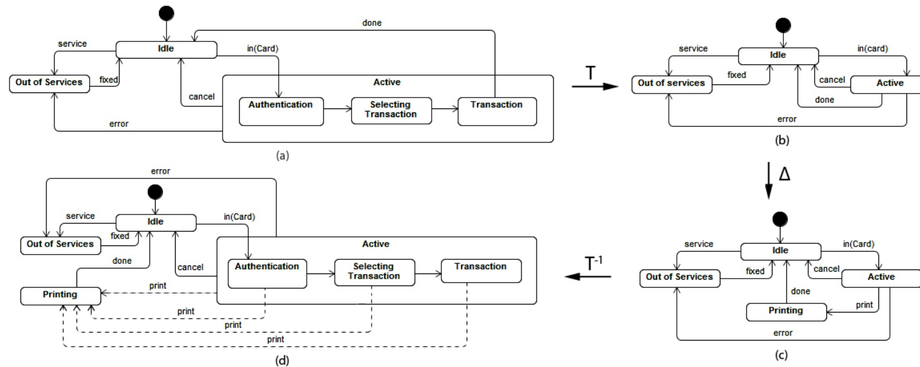


**Fig. 1.** Collapse/expand state diagrams in a round-trip process

**Scenario.** Let us consider a typical round-trip problem based on the *Collapse/Expand State Diagrams* benchmark [5]. In particular, starting from a hierarchical state diagram (involving some nesting) as the one reported in Fig. 1(a), the bidirectional transformation yields a flat state machine as provided in Fig. 1(b). A fundamental requirement of the transformation prescribes that manual modifications on the target model must be back propagated to the source model. For instance, suppose that the designer modifies the flattened machine in Fig. 1(b) to produce the model in Fig. 1(c) by:

- adding the new state `Printing`,

– adding the transition `print` that associates state `Active` to the latter, and finally
– modifying the source of the transition `done` from the state `Active` to the state `Printing`.

The expected transformation is clearly non-injective (as different hierarchical machines can be flattened to the same model). In addition, such a model refinement gives place to an interesting situation, i.e., more than one model is admissible (see dotted edges in Fig. 1(d)).

**Implementation.** The *HSM2SM* bidirectional transformation, which relates hierarchical and flat state machines, has been implemented by means of JTL: a constraint-based model transformation language specifically tailored to support bidirectionality. It adopts a QVT-R[1] like syntax and allows a declarative specification of relationships between MOF models. The semantics is given in terms of Answer Set Programming (ASP) [15], which 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[2] finds and generates, in a single execution, all the possible models which are consistent with the transformation rules by a deductive process. The JTL environment has been implemented as a set of plug-ins for the Eclipse framework and mainly exploits EMF[3].

```
1 transformation hsm2sm(source : HSM, target : SM) {
2  ...
3  top relation Transition2Transition {
4   enforce domain source sourceTrans: HSM::Transition{
5    owningStateMachine = sourceSM: HSM::StateMachine { },
6   };
7   enforce domain target targetTrans: SM::Transition{
8    owningStateMachine = targetSM: SM::StateMachine { },
9   };
10  when {...}
11  where {...}
12 }
13 relation TransitionSource2TransitionSource {
14  enforce domain source sourceTrans: HSM::Transition {
15   source = sourceState : HSM::State { }
16  };
17  enforce domain target targetTrans: SM::Transition {
18   source = targetState : SM::State { }
19  };
20  when {
21   State2State(sourceState, targetState) and
22   sourceState.owningCompositeState.oclIsUndefined();
23  }
24 }
25 relation TransitionSourceComposite2TransitionSource {
26  enforce domain source sourceTrans: HSM::Transition {
27   source = sourceState : HSM::CompositeState { }
28  };
29  enforce domain target targetTrans: SM::Transition {
30   source = targetState : SM::State { }
31  };
32  when { CompositeState2State(sourceState, targetState); }
33 }  ...
```

**Listing 1.1.** A fragment of the HSM2SM transformation in JTL

[1] http://www.omg.org/spec/QVT/1.1/
[2] http://www.dlvsystem.com/
[3] http://www.eclipse.org/modeling/emf/

A fragment of the *HSM2SM* transformation is illustrated in List. 1.1. It consists of a number of *relations* defined by the two involved *domains*. In particular, the following relations are reported:

- *Transition2Transition* which relates transitions of the hierarchical metamodel and transitions of the flat metamodel,
- *TransitionSource2TransitionSource* which relates source states of transitions of the hierarchical metamodel and the corresponding source states of transitions of the flat metamodel, and finally
- *TransitionSourceComposite2TransitionSource* which relates source composite states of transitions of the hierarchical metamodel and correspondent source states of transitions of the flat metamodel[4].

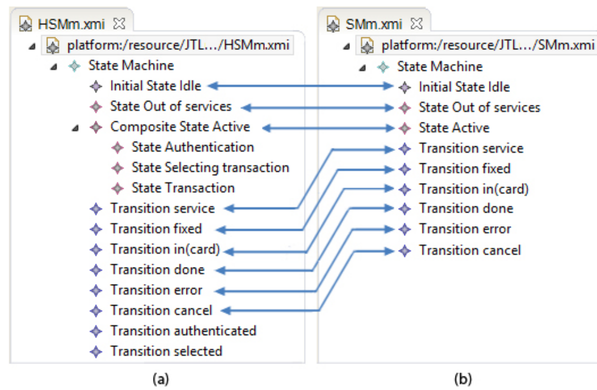


**Fig. 2.** The HSM model and the correspondent SM model

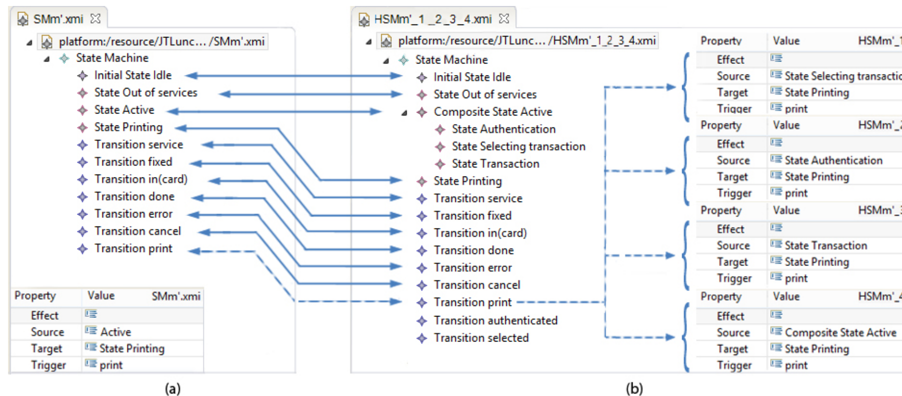


**Fig. 3.** The modified SM model and the correspondent HSM models

The forward application of the transformation is illustrated in Fig. 2, where the model *HSMm* on the left-hand side is mapped to *SMm* in the right-hand side. As aforementioned the transformation is non-injective. The back propagation of the changes

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

showed in Fig. 1(c) therefore gives place to the following situation: the newly added transition `print` can be equally mapped to each of the nested states within `Active` as well as to the container state itself, as in Fig. 1(d). In particular, the modified target model *SMm'* in Fig. 3(a) is mapped back to the source models $HSMm'_1, ..., HSMm'_4$ in Fig. 3(b). For example, as visible in the property of the `print` transition, $HSMm'_4$ represents the case in which the transition target goes to the composite state `Active`.

Such non-determinism can still be resolved by accommodating in the transformation the prescription that any new edges in a target model should be mapped to an edge in corresponding source model, such that it *always* refers to the same kind of state, e.g., the container state. This would definitely make the transformation deterministic. However, when the solution cannot be singled out in such a general way, the decision must be left to the modeler in later stages. The potential information erosion have a negative impact on software cost and quality [19]. Thus, designers dealing with model uncertainty need to be supported with suitable mechanism and tools in order to avoid effect of having multiple design alternatives.

## 3 Uncertainty in modeling

In software engineering decisions have to be made at different stages. Despite the nature of software means making these decisions with absolute confidence, uncertainty appears everywhere [22]. Typically, it occurs when the designer does not have complete, consistent and accurate information required to make a decision during software development. Introducing uncertainty in modeling processes means that, rather than having a single model, we actually have a set of possible models and we are not sure which is the correct one [12]. Thus, handling uncertainty requires the modeler to use this set whenever an individual model would be used. In addition, managing a set of models explicitly is impractical as its size might be quite large. On the other hand, if uncertainty is ignored and one particular possible model is prematurely chosen, we risk having incorrect information in the model. Recently, an approach [12, 14] has been proposed to cope with different aspects of this problem. In particular, the concept of *partial model* has been given in terms of graph theory to capture uncertainty in models. In essence, by means of first-order logic annotations *points of uncertainty* can be introduced in the model, each denoting a possible *concretization*, i.e., a model where the uncertainty is resolved.

Non-bijective model transformations are strictly related to uncertainty, that is introduced during the transformation writing but it is often evident only after the execution, when more than one models may be generated. Especially when the set of generated models is large, designers need to be supported by suitable mechanisms and tools able to manage uncertainty. For this reasons, our aim is to provide the designer a tool for represent the different models into a new one that contains all generated alternatives which abstracts from the calculation method and permits to harness the potential offered by generic modeling platforms such as Eclipse/EMF ([2]]). Furthermore, it represents the starting point for extending a language like JTL semantics and its transformation engine towards an uncertainty-aware solution, capable of dealing with the intrinsic non-determinism of non-bijective transformation in terms of uncertain or partial models.

## 4 A metamodel-independent approach to uncertainty

Uncertainty as it is known in literature (e.g., [12]) does not have a characterization in terms of metamodels. It is mainly based on annotations which can be processed by tools which are outside, for instance, the EMF ecosystem. In this section, we introduce a metamodel-independent approach to uncertainty representation, i.e., starting from a *base* metamodel $M$ we are interested to understand what are the characteristics of the corresponding metamodel with uncertainty $U(M)$ and how constructively define it.

Since, it has to be used in modeling environment and must be processed by means of automated transformation, in according to our view, we identified a number of natural properties that representation technique should have, as described below

- *model-based*, a set of models representing different alternatives must be represented with a model with uncertainty enabling a range of operations, including analysis or manipulations during the decision process;
- *minimality*, a model with uncertainty is a concise representation of all the alternative solutions; it should not contain any other information besides what needed for representing both the common elements and alternative elements, i.e. alternative designed choices grouped by a point of uncertainty;
- *metamodel-independence*, the metamodel must be agnostic of the base metamodel, i.e., it must be defined in a parametric way such that the definition procedure can be applied in an automated way to any metamodel;
- *interoperability*, each model containing uncertainty must be applicable an unfolding operation, such that whenever applied to it returns all the correspondent concretizations models or the specific concretization selected by the designer.

Starting from these requirements, an automated procedure $U$ : *Ecore* $\rightarrow$ *Ecore* is proposed. The transformation is written in ATL and takes a metamodel $M$ and returns the corresponding metamodel with uncertainty $U(M)$ as described in the rest of the section.

### 4.1 The uncertainty metamodel

The metamodel with uncertainty is obtained by extending the base metamodel with given connectives to represent the multiple outcomes of a transformation (as showed in Sect. 2). These connectives denote points of uncertainty where different model element are attached. Moreover, such points of uncertainty are traceable in order to ease the traversal of the whole solution space and permit the identification of specific concretizations.

As an example, let us consider *HSM*, the metamodel of the hierarchical state machines given in Fig. 4. Then the corresponding metamodel with uncertainty $U(\textit{HSM})$ illustrated in Fig. 5 can be automatically obtained as follows:

- the abstract metaclass `TracedClass` with attributes `trace` and `ref` is s added to $U(\textit{HSM})$;
- for each metaclass *c* in *HSM*, such that it is non-abstract and does not specialize other metaclasses, *i)* a corresponding metaclass *uc* is created in $U(\textit{HSM})$ such that *uc* specializes *c*, and *ii)* *c* is generalized by `TracedClass`;
- each metaclass *uc* is composed with *c*, enabling the representation of a point of uncertainty and its alternatives;

– the cardinality of attributes and references derived from *HSM* are relaxed and made optional in $U(HSM)$ in order to permit to express uncertainty also over them.

In particular, the metaclasses `UStateMachine`, `UState` and `UTransition` in $U(HSM)$ derive from `StateMachine`, `State` and `Transition` in *HSM*, whereas the latter ones are generalized by `TracedClass`. The scope of this abstract class is to maintain information about the relationships between the points of uncertainty and the correspondent own alternatives in the concretization models.

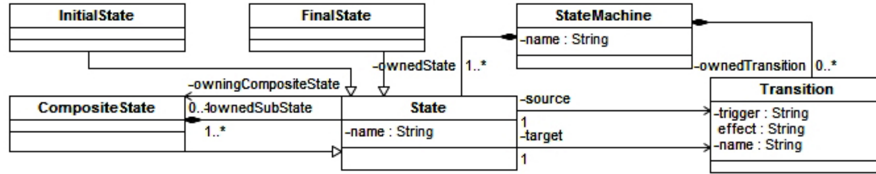


**Fig. 4.** The *HSM* metamodel

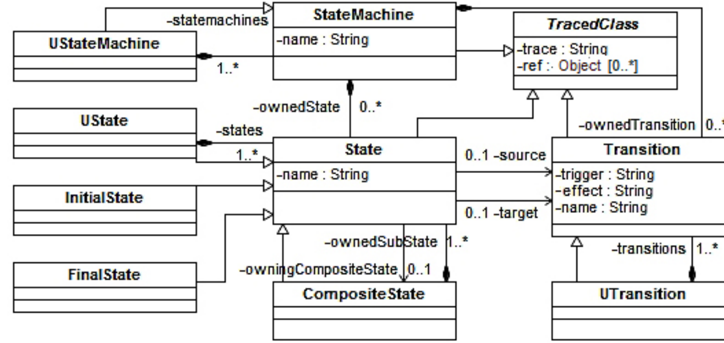


**Fig. 5.** The $U(HSM)$ metamodel

As said, the above procedure is implemented as an endogenous model transformation in ATL. For the sake of brevity, only an excerpt of the transformation is presented in Listing 1.2[5] containing solely those rules which build the specific constructions of the uncertainty metamodel. More in detail, the rule `EClass2UEClass` (lines 4-20) *a)* propagates base metaclasses (and their associations) which are not abstract and do not have non-abstract ancestors (lines 5-6) and *b)* for each of them generates a corresponding uncertainty metaclass (lines 11-14), as in Fig. 5 where the metaclass `UState` is generated and composed with `State`. Moreover, the target pattern (lines 7-20) is composed of a set of elements, each of them specifies a target type from the target metamodel and a set of bindings. In particular, the element `t` (lines 7-10) copies the metaclass `s` in the target metamodel; the element `u` of the target pattern (lines 11-14) generates uncertainty metaclass as specialization of the matched source class `s`; and finally, the reference `r` is created as a structural feature of the element `u` in order to refer alternative elements contained in `u`.

```
1 module MM2UMM;
2 create OUT : UMM from IN : MM;
```

[5] The *MM2UMM* transformation implementation is available at http://jtl.di.univaq.it/

```
3 [...]
4 rule EClass2UEClass {
5     from s : MM!EClass ((thisModule.inElements->includes(s)) and
6         ((s.eSuperTypes->size()=0) or (s."abstract"='false')))
7     to t : UMM!EClass (
8         name <- s.name,
9         eSuperTypes <- s.eSuperTypes->append(thisModule.traceableMetaclass),
10        ...),
11    u : UMM!EClass (
12        name <- 'U'+s.name,
13        eReferences <- Sequence{}->append(r),
14        eSuperTypes <- Sequence{}->append(s)),
15    r:UMM!EReference(
16        name <- s.name + 's',
17        containment <- true,
18        lowerBound <- 1,
19        upperBound <- -1)
20 }
21 [...]
```

**Listing 1.2.** A fragment of the MM2UMM transformation

To better understand how a point of uncertainty is realized, please consider the alternative solutions in the right-hand side of Fig. 3 and how they are denoted by the corresponding point of uncertainty illustrated in Fig. 5. In particular, the alternative transitions are collected in a point of uncertainty (`UTransition`) which contains the transitions `print` targeting each one of the nested states within `Active` as well as to the composite state itself.

It is important to notice that models with uncertainty may be an *over-approximation* 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 solution space. For instance, in the scenario in Fig. 1, probably only one `print` transition can exist in the final model. However, the generated model with uncertainty admits models with multiple `print` transitions giving place to more concretizations than those expected. Therefore, besides the models with uncertainty it is important to generate also those constraints which limit the solution to the admissible concretization, in our case, in order to avoid multiple `print` transitions, the model with uncertainty in Fig. 5 is augmented with a constraint that reduces the concretizations to cases with one `print` transition only. According to the uncertainty metamodel described in Sect. 4, each metaclass provide an attribute `ref` to maintain the reference to the corresponding concretization(s).

# 5 Related work

Uncertainty is one of the factors prevalent within contexts as requirements engineering [7], software processes [16] and adaptive systems [18]. Uncertainty management has been studied in many works, often with the aim to express and represent it in models. In [12] *partial model*s are introduced to allow designers to 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 [13] 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.

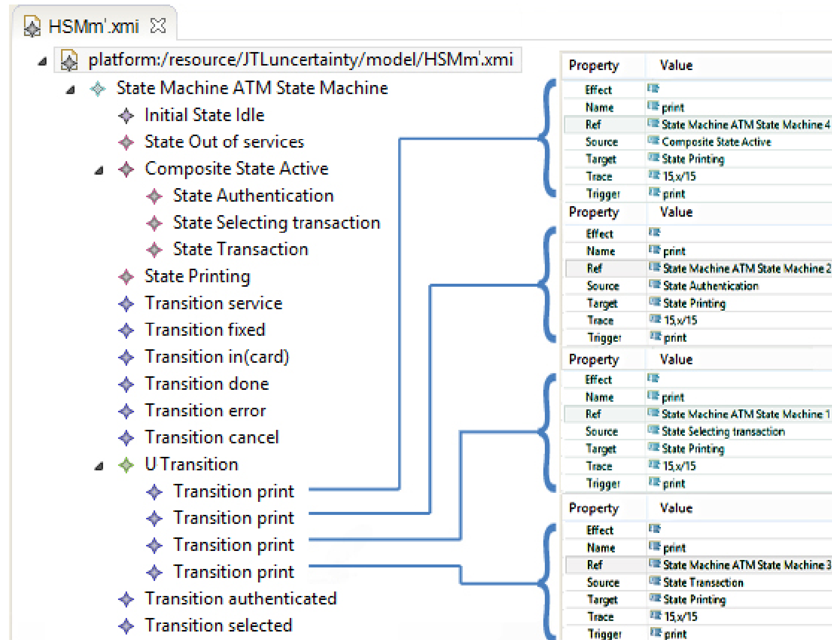


**Fig. 6.** UHSMm model

As discussed in this paper, modelers may need to encode ambiguities in their model transformation and obtain multiple design alternatives in order to choose among them. In contrast with this requirement, most existing bidirectional model transformation languages deal with non-determinism by requiring designers to write non-ambiguous mappings in order to obtain a deterministic result [1, 23, 6] . The ability to deduce and generate all the possible solutions of an uncertain transformation has been achieved by few approaches, including JTL [8, 10]. In such case, may be useful to manage non-determinism during the design process in order to detect ambiguities and support designers in solving non-determinism in their specification as faced in [9].

## 6 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. In this paper, we tackle the problem of non-determinism in bidirectional transformations focusing on the concept of uncertainty, which represent one of the prevalent factors within software engineering. When modelers 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. In this work we have made an attempt to help designers to give an uniform characterization of the solution in terms of models with uncertainty as already known in literature.

# 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. *SOSYM*, 2007.
2. J. Bézivin, F. Jouault, P. Rosenthal, and P. Valduriez. Modeling in the Large and Modeling in the Small. In *Procs of European MDA Workshops*, 2004.
3. G. Callow and R. Kalawsky. A Satisficing Bi-Directional Model Transformation Engine using Mixed Integer Linear Programming. *JOT*, 12(1):1: 1–43, 2013.
4. A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *SLE10*, pages 183–202, 2010.
5. 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*.
6. Z. Diskin, Y. Xiong, and K. Czarnecki. From state- to delta-based bidirectional model transformations. pages 61–76, 2010.
7. C. Ebert and J. D. Man. Requirements uncertainty: influencing factors and concrete improvements. In *Procs. of ICSE*, pages 553–560. ACM Press, 2005.
8. 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.
9. R. Eramo, R. Marinelli, A. Pierantonio, and G. Rosa. Towards Analyzing Non-Determinism in Bidirectional Transformations. In *Procs. of AMT 2014*, 2014.
10. R. Eramo, A. Pierantonio, J. R. Romero, and A. Vallecillo. Change management in multi-viewpoint system using asp. In *EDOCW08*, pages 433–440. IEEE Computer Society, 2008.
11. R. Eramo, A. Pierantonio, and G. Rosa. Uncertainty in bidirectional transformations. In *Procs. of MiSE 2014*, 2014.
12. M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *ICSE*, pages 573–583, 2012.
13. M. Famelis, R. Salay, A. D. Sandro, and M. Chechik. Transformation of models containing uncertainty. In *MoDELS*, pages 673–689, 2013.
14. M. Famelis and S. Santosa. Mav-vis: A notation for model uncertainty. In *MiSE*, 2013.
15. 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.
16. 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.
17. N. Macedo and A. Cunha. Implementing QVT-R Bidirectional Model Transformations Using Alloy. In *FASE*, pages 297–311, 2013.
18. 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, 2010.
19. B. Schätz, F. Hölzl, and T. Lundkvist. Design-space exploration through constraint-based model-transformation. In *ECBS*, pages 173–182, 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. A. Sillitti, M. Ceschi, B. Russo, and G. Succi. Managing uncertainty in requirements: A survey in documentation-driven and agile companies. In *IEEE METRICS*, page 17. IEEE Computer Society, 2005.
23. P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *SOSYM*, 8, 2009.
24. S. Witkop. MDA users' requirement for QVT transformations. In *OMG doc 05-02-04*, 2005.