Transformation of Models Containing Uncertainty

Michalis Famelis, Rick Salay, Alessio Di Sandro and Marsha Chechik

University of Toronto
Toronto, Canada
{famelis,rsalay,adisandro,chechik}@cs.toronto.edu

Abstract. Model transformation techniques typically operate under the assumption that models do not contain uncertainty. In the presence of uncertainty, this forces modelers to either postpone working or to artificially remove it, with negative impacts on software cost and quality. Instead, we propose a technique to adapt existing model transformations so that they can be applied to models even if they contain uncertainty, thus enabling the use of transformations earlier. Building on earlier work, we show how to adapt graph rewrite-based model transformations to correctly operate on May uncertainty, a technique that allows explicit uncertainty to be expressed in any modeling language. We evaluate our approach on the classic Object-Relational Mapping use case, experimenting with models of varying levels of uncertainty.

1 Introduction

Model Driven Engineering (MDE) promises to accelerate and improve the quality of software development: software is described using high-level models which are easy to reason with. These models are then transformed into lower-level designs through a series of model transformations. Finally, low-level designs are used for effective code generation.

One of the factors prevalent within software engineering is *model uncertainty* which exists whenever a modeler is unsure about the information in the model. Uncertainty stems from a variety of causes including stakeholder conflicts [16], incomplete information [28], alternative design decisions [25], etc. Existing MDE solutions do not handle models with uncertainty. So when uncertainty is unresolved, the modeler should either delay the application of transformations until more information becomes available, or make premature resolutions in order to apply transformations, thus creating a risk that these resolutions are incorrect. In either case, uncertainty diminishes the benefits of MDE.

In this paper, we propose an approach that allows applying *existing* transformations to models containing uncertainty. The essence of the approach involves automatically modifying – "lifting" – transformations so they operate on models with uncertainty and correctly transform both the content of the model and the uncertainty about it. As a result of our approach, transformations can be applied early in the model development lifecycle, tolerating the uncertainty and allowing

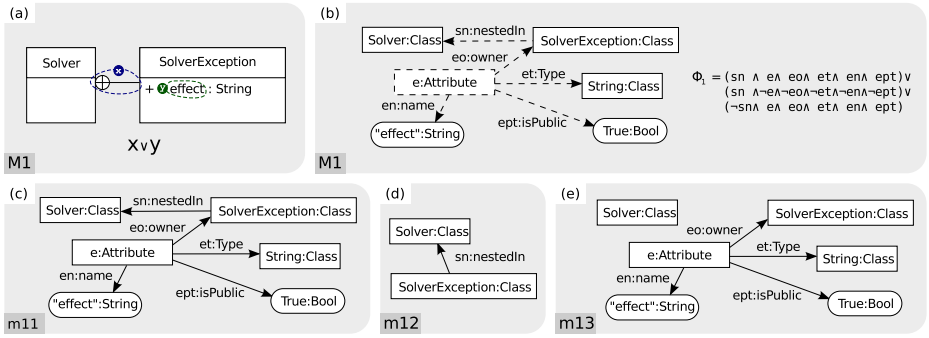


Fig. 1. (a) May model M_1, showing points of uncertainty. (b) M_1 as a typed graph (UML abstract syntax). (c-e) Concretizations m_{11}, m_{12}, m_{13} of M_1.

modelers to defer its resolution until extra information is available. This eliminates the need to delay transformation application and removes the pressure to potentially compromise model quality by resolving the uncertainty prematurely.

Uncertainty has been studied in different contexts including requirements engineering [6], adaptive systems [21] and software processes [13] but there has been little work specifically on *model* uncertainty. To address this gap, we have proposed a language-independent method of expressing uncertainty within models [10], through *May models*. May models allow a modeler to specify whether an element *should* be present, or its presence is *unknown*. Optional *May formulas* specify constraints over presence/absense of elements, in order to disallow infeasible or undesired uncertainty resolutions.

In this paper, we develop an approach for automatic "lifting" of transformations *specified as graph rewrite rules* so they *apply to models containing May uncertainty*. Specifically, the paper makes the following contributions:

1. an approach for transforming models that contain uncertainty, centered around the notion of lifting;
2. an automated method for creating lifted versions of existing transformation rules, so that they apply to May models;
3. an application of the approach to the classic Object-Relational Mapping (ORM) problem to assess its feasibility.

Relation to previous work. We first introduced the notion of lifting model transformations in [11]. In that approach, lifting is accomplished purely in propositional logic, via the use of *transfer predicates*. Rules are first turned into "template predicates" which, given a match in the input model, are instantiated with the propositional variables of the input model. This approach has a number of problems: (a) Transfer predicates need to be constructed ad-hoc, separately for each rule. On the other hand, the lifting approach proposed in this paper can be used for *arbitrary* rules. (b) Instantiation of the templates requires a-priori knowledge of the vocabulary of the input model, making it awkward to handle expanding/contracting vocabularies via additive/deleting rules. Instead, lifting is not dependent on the input model or the matching site. (c) The transfer

predicates approach cannot handle NACs. (d) The May models resulting from applying the transfer predicates approach are expressed as propositional formulas over the set of both the input and the output model variables: in effect, the new model was expressed as a delta from the old. Thus, obtaining the new model required quantifying the old variables out – a computationally expensive process. (e) The transfer predicate approach requires a separate testing step to verify that the transformation was applied correctly. Moreover, testing entails enumerating *all* concretizations! Instead, the lifting presented here is guaranteed to be correct by construction.

In [10], we studied a different form of May model transformation: uncertainty-reducing refinement. The goal of this transformation is to reduce the number of possible concretizations of a given model with uncertainty, by specifying additional information. Work in [19, 18] expanded the scope of the problem to additional uncertainty types, comprising the *MAVO* uncertainty framework described in [20]. The objective of this work is different: lifted transformations do not change the level of uncertainty by removing concretizations. Moreover, we aim to adapt classical transformations to May models instead of developing and checking transformations developed specifically for models with uncertainty.

The rest of this paper is organized as follows: We introduce a motivating example in Sec. 2. In Sec. 3, we give the necessary background. We describe the lifting process in Sec. 4 and evaluate it in Section 5. After comparing our approach with related work in Sec. 6, we conclude in Sec. 7 with a summary and a discussion of follow-on research.

2 Motivating Example

In this section, we introduce a running example to motivate and illustrate the key points of our approach. Suppose a modeler is creating a UML class diagram for an automated reasoning engine. The modeler has decided that there should exist a class `Solver` which throws exceptions, objects of type `SolverException`, whenever it reaches an error state. However, the modeler has yet to make the following design decisions: (a) whether `SolverException` should be an inner class of `Solver`, and (b) whether `SolverException` should have a `String` attribute called `effect` that would record an estimation of the effect of the exception on the reasoning process. In addition, the modeler expects that at least one of these features will be present in her model.

The resulting UML class diagram with uncertainty is encoded as a May model M_1 in Fig. 1(a). In this model, "$\oplus$" is the UML symbol for a "nested class" and used here to indicate that `SolverException` is an inner class of `Solver`. The syntax for capturing uncertainty in M_1 is described in [14]. This model has two *points of uncertainty* – the relationship between the classes `Solver` and `SolverException` (denoted by `x`) and the presence of the `effect` attribute in class `SolverException` (denoted by `y`). *Maybe* elements that make up each point of uncertainty are enclosed in dashed ellipses. To better illustrate the details of each point of uncertainty, we show M_1 as a type graph (a simplified version of the

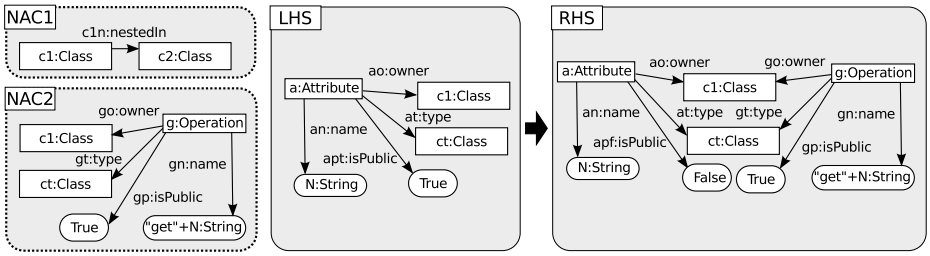


Fig. 2. Transformation rule R_{EV} for doing the *Encapsulate Variable* refactoring.

UML abstract syntax) in Fig. 1(b). The node `e` and the edges `sn`, `eo`, `et`, `en`, `ept` are annotated with Maybe and thus indicated using dashed lines. An additional *May formula* Φ_1, also shown in Fig. 1(b), constrains the possible combinations of Maybe elements, defining the possible ways in which uncertainty can be resolved. Specifically, each Maybe element is represented by a propositional variable of the same name in the May formula. Thus, allowable configurations of the May model correspond to valuations of the variables that satisfy the May formula. In our scenario, the modeler's uncertainty can be resolved in one of three possible ways, corresponding to the models m_{11}, m_{12}, m_{13} shown in Figs. 1(c-e), respectively. These models are called *concretizations* of M_1.

Assume that the modeler notices that her model has an anti-pattern, namely, that the attribute `effect` is `public`. She decides that, unless `SolverException` is an inner class, `effect` should be made `private` for security reasons, and be accessed through a getter method. This can be accomplished by performing the *Encapsulate Variable* refactoring [3]. A generic method for implementing this refactoring using graph transformations was described by Mens et al. [15]. A simplified version of this rule, called R_{EV}, is shown in Fig. 2. The left-hand side (LHS) of the rule matches a node a (and its associated edges such as ao:`owner`) that represents a public attribute . The right-hand side (RHS) makes it private (by deleting the `isPublic` edge apt from a to `True` and adding a new `isPublic` edge apf from a to `False`). It also creates a public getter operation ge and its associated edges. In addition, the rule has two negative application conditions (NACs), i.e., conditions under which the rule should not be applied. These are: NAC_1, specifying the case when the class containing the public attribute is an inner class, and NAC_2, specifying the case when the class already has a getter.

Rule R_{EV} cannot be directly applied to the May model M_1 because it contains uncertainty. Our goal is thus to create its "lifted" version, $\mathcal{R}_{EV}$ that can be applied directly to M_1. The intuition behind such a lifting is as follows [11]: take the three concretizations, m_{11}, m_{12}, m_{13}, of M_1; apply R_{EV} to each of them, resulting in models m_{21}, $_{22}$, m_{23} in Fig. 3(a-c); represent the resulting models as a May model M_2 in Fig. 3(d). That is, applying the lifted rule to a May model should be equivalent to a representation of the result of applying the original rule to each of the concretizations of the May model. So, applying the lifted version $\mathcal{R}_{EV}$ of R_{EV} to M_1 should produce the May model M_2 directly, without having to produce and transform individual concretizations. In this paper, we

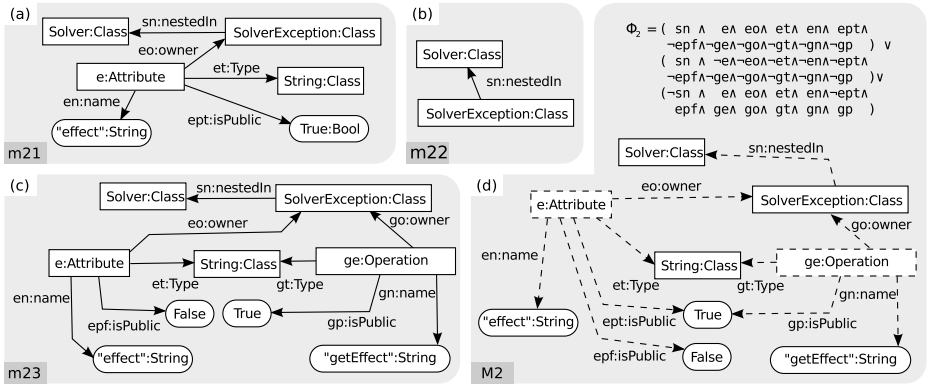


Fig. 3. (a-c) Classical models resulting from applying R_{EV} to the classical models in Fig. 1(c-e). (d) The result of applying the lifted version $\mathcal{R}_{EV}$ to M_1 directly: the May model M_2.

use this example to describe our technique for lifting transformations expressed as graph rewrite rules to correctly handle May models.

3 Background

In this section, we provide the background necessary for the rest of the paper and fix the notation.

May Models. The formal definition and semantics of May models is given in [12]. In this section, we give an informal definition and illustrate it using the motivating example.

Definition 1 (May model) *A* May model *M is a tuple $\langle G, ann, \Phi_M \rangle$, where G is a typed graph called a* base graph*, ann is a function that annotates a subset S_M of elements of G with* Maybe*, and Φ_M is the* May formula*. The tuple $\langle G, ann \rangle$ representing the* Maybe*-annotated typed graph G is called the* May graph*. S_M is denotes the set of all* Maybe *elements of M.*

The base graph is typed by a metamodel, represented by a *type graph*. A simplified type graph for class diagrams is shown in Fig. 4. M_1 is shown as an instance of this type graph in Fig. 1(b) Annotating an element with Maybe indicates the uncertainty of the modeler about whether that element should be part of the model or not. In Fig. 1(b), Maybe-annotated elements such as the attribute node e are shown with dashed lines.

Each Maybe element is represented by a propositional variable which expresses the proposition *"the element is part of the model"*. The May formula is expressed over this vocabulary of variables. Allowable configurations of Maybe elements are thus specified by the satisfying assignments of the May formula.

Definition 2 (Concretization) *A* concretization *of a May model $\langle G, ann, \Phi_M \rangle$ is a classical model derived from M by assigning each propositional variable for*

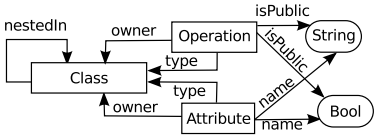


Fig. 4. Simple type graph for class diagrams.

a Maybe *element of M to either True or False, such that Φ_M is satisfied. The set of all concretizations of a May model M is denoted by $[M]$.*

Thus, a May formula ensures that the corresponding May model is an *exact* representation of a set of classical models.

$[M_1] = \{m_{11}, m_{12}, m_{13}\}$ where the classical models m_{11}, m_{12}, m_{13} are shown in Fig. 1(c-e). Each of them represents a case where all uncertainty is resolved, i.e., all variables corresponding to Maybe elements have been set to either True or False. For example, model m_{13} in Fig. 1(e) can be obtained by satisfying the last disjunctive clause of Φ_1, i.e., by setting `sn` to False and `e`, `eo`, etc. to True.

For example, consider a May model M_1' in which the class `Solver` is also annotated with Maybe, but whose May formula is changed so that each clause contains the non-negated term `Solver`. Even though `Solver` is annotated with Maybe in M_1', all concretizations of M_1' must contain it and thus $[M_1] = [M_1']$. We call models that have the same set of concretizations *equivalent*. A May model M is said to be in the *graphical reduced form* (GRF) iff an element is annotated with Maybe in the May graph iff it is not common to all of M's concretizations. In [12] we give an algorithm for transforming any May model to a GRF equivalent model. M_1 is given in GRF.

Model Transformations. We focus on *graph transformations* [9]. Such transformations apply to models that do not contain uncertainty, e.g., m_{11} in Fig. 1(c). They are implemented by executing a set of graphical rules defined as follows:

Definition 3 (Transformation rule) *A* transformation rule *R is a tuple $R = \langle\{NAC\}, LHS, RHS\rangle$, where the typed graphs LHS and RHS are respectively called the* left-hand *and the* right-hand *sides of the rule, and $\{NAC\}$ represents a (potentially empty) set of typed graphs, called* negative application conditions.

We show the RHS and LHS of the rule R_{EV} in Fig. 2. In addition, R_{EV} contains two NACs (NAC_1 and NAC_2), indicated by a dotted border.

The LHS, RHS and NACs of a rule consist of different *parts*, i.e., sets of model elements which do not necessarily form proper graphs. These parts play different roles during the rule application:

$\mathbf{C}^r$: The set of model elements that are present both in the LHS and the RHS, i.e., remain unaffected by the rule.

$\mathbf{D}^r$: The set of elements in the LHS that are absent in the RHS, i.e., deleted by the rule.

$\mathbf{A}^r$: The set of elements present in the RHS but absent in the LHS, i.e., added by the rule.

$\mathbf{N}^r$: The set of elements present in any NAC, excluding those included in $\mathbf{C}^r$.

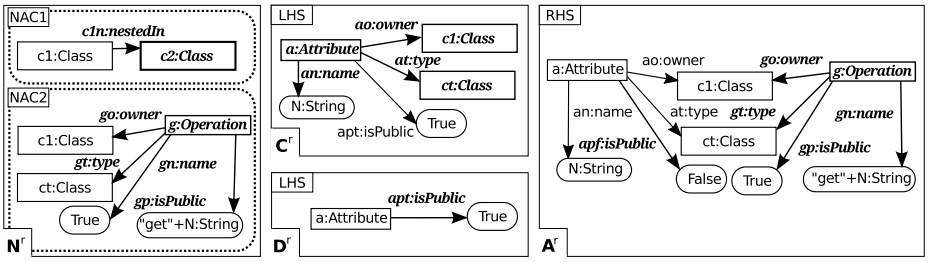


Fig. 5. Parts of the rule R_{EV}. Each rule part contains only those elements whose label appears ***in bold serif font***[1].

The parts of the example rule R_{EV} from Fig. 2 are shown in Fig. 5. Specifically, $\mathbf{C}^r$ is the set[1] $\{a, ao, an, at, c1, ct\}$, $\mathbf{D}^r$ is the (unary) set $\{apt\}$, $\mathbf{A}^r$ is $\{g, go, gn,\ gp, gt, apf\}$, and $\mathbf{N}^r$ is $\{g, gn, go, gt, c1n, c2\}$.

A rule R is *applied* to a model m by finding a *matching site* of its LHS in m:

Definition 4 (Matching site) *A* matching site *of a transformation rule R in a model m is a tuple $K = \langle\overline{\mathbf{N}}$,$\mathbf{C}$,$\mathbf{D}\rangle$, where $\mathbf{C}$ and $\mathbf{D}$ are matches of the parts $\mathbf{C}^r$ and $\mathbf{D}^r$ of the LHS of R in m, and $\overline{\mathbf{N}}$ is the set of all matches of NACs in m that are anchored at the matches $\mathbf{C}$ and $\mathbf{D}$.*

For example, a matching site K_1 for the rule R_{EV} in the model m_{11} in Fig. 1(c) is $\langle\mathbf{C}_1, \overline{\mathbf{N}}_1, \mathbf{D}_1\rangle$, where $\mathbf{C}_1 = \{$`e`, `eo`, `en`, `et`, `SolverException`, `String`$\}$, $\overline{\mathbf{N}}_1 = \{\{$`Solver`, `sn`$\}\}$, and $\mathbf{D}_1 = \{$`ept`$\}$.

In the above definition, $\overline{\mathbf{N}}$ denotes the set of all matches within m of the NACs of R. given the match of $\mathbf{C}^r$ and $\mathbf{D}^r$. If the same NAC can match multiple ways, then all of them are included in $\overline{\mathbf{N}}$ as separate matches. For example, if the model in Fig. 1(c) had another class `Solver2` that also nested `SolverException` via an edge `sn2`, then $\overline{\mathbf{N}}$ would contain two matches for NAC_1: $\overline{\mathbf{N}}$= $\{\{$`Solver`,`sn`$\}$,$\{$`Solver2`,`sn2`$\}\}$. The set of matching sites define the places in the model m where the rule can potentially be applied.

Definition 5 (Applicability condition) *Given a transformation rule R, a model m, and matching site $K = \langle\overline{\mathbf{N}}$, $\mathbf{C}$, $\mathbf{D}\rangle$, the rule R is* applicable at K *iff $\overline{\mathbf{N}}$ is empty*[2].

The above definition ensures that the rule can only be applied at a given site if no NAC matches. For R_{EV}, the matching site K_1 in m_{11} does not satisfy the applicability condition as $\overline{\mathbf{N}}_1 \neq \emptyset$. On the other hand, the model m_{13} in Fig. 1(e) contains a matching site $K_2 = \langle\emptyset, \mathbf{C}_1, \mathbf{D}_1\rangle$, which does satisfy this condition. Then, the rule can be applied:

[1] Nodes that represent values (e.g., boolean True, the string N, etc.) are also considered to be part of $\mathbf{C}^r$ but are omitted here for brevity.

[2] The theory of graph transformation requires some additional formal preconditions, most notably the *gluing condition* [9]. These are not discussed here for brevity.

Definition 6 (Rule application) *Given a transformation rule R, a model m, and a matching site K in m for which the rule applicability condition is satisfied, rule R is* applied*, producing a model m', by removing $\mathbf{D}$ from m and adding $\mathbf{A}$, where $\mathbf{A}$ is a match of the part $\mathbf{A}^r$ of R in m. Rule application is denoted as $m \overset{R}{\Longrightarrow} m'$.*

Applying R_{EV} to m_{13} at K_2 thus requires the deletion of the element `ept` because it is contained in $\mathbf{D}$, and the addition of new elements according to $\mathbf{A}^r$. The resulting model m_{23} is shown in Fig. 3(c), where $\mathbf{A}$ is the set {`ge`, `go`, `gn`, `gp`, `gt`, `epf`}.

We refer to rules such as the ones described above as "classical" to differentiate them from their "lifted" counterparts which can be applied to May models.

4 Lifting Transformations

In this section, we describe the process of lifting a transformation rule to apply to May models. A classical rule R adapted to apply to May models is called *lifted* and is denoted by $\mathcal{R}$.

May models are intended to be exact representations of sets of models and lifted transformations should preserve this. Therefore, applying a lifted transformation rule $\mathcal{R}$ to a May model M_{in} should be equivalent to applying its classical version R to each of the concretizations of M_{in} and building a May model from the result. We refer to this principle, defined in [11], as the *Correctness Criterion* for lifting transformations and define it formally below.

Definition 7 *Let a rule R, a May model M_{in} with a set of concretizations $[M_{in}] = \{m^1_{in}, \ldots, m^n_{in}\}$, and the set $U = \{m^i_{out} \mid \forall m^i_{in} \in [M_{in}] \cdot m^i_{in} \overset{R}{\Longrightarrow} m^i_{out}\}$ be given. $\mathcal{R}$ is a* correct lifting *of R iff for any production $M_{in} \overset{\mathcal{R}}{\Longrightarrow} M_{out}$, the set of concretizations of the resulting May model M_{out} satisfies the condition $[M_{out}] = U$.*

In the motivating example, we aim to compute a lifted version $\mathcal{R}_{EV}$ of R_{EV} s.t. for the May models M_1 and M_2 in Figs. 1(a) and 3(a), $M_1 \overset{\mathcal{R}_{EV}}{\Longrightarrow} M_2$.

In traditional rule application during graph transformation, it is sufficient to find a graph match of the LHS of the rule and then check whether the NACs are applicable. However, a May model also has a propositional component, the May formula which constrains the possible combinations of Maybe elements. Thus, doing the graphical match for the May graph is not sufficient to guarantee correctness and needs to be augmented with manipulation of the May, to ensure that the appropriate concretizations get transformed. We illustrate both parts of the transformation on our running example in Sec. 4.1 and then generalize in Sec. 4.2. In Sec. 4.3, we prove correctness of this approach.

4.1 Lifting example

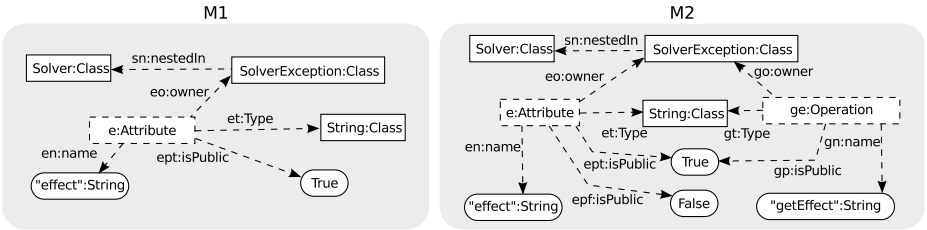
We illustrate the transformation of the graph and the formula using our running example where the rule R_{EV} in Fig. 2 is applied to the May model M_1, shown in Fig. 1(b), to produce M_2 in Fig. 3(d). Fig. 6 summarizes the application of this rule for the single existing matching site, showing the May graphs and the truth tables of the May formulas Φ_1 and Φ_2 of M_1 and M_2, respectively. Each column of the truth tables is a Maybe element. Each row corresponds to an allowable configuration of Maybe elements, denoted by 1, and thus defines a concretization. The truth tables also show which Maybe elements are matched by each of the rule's parts. For example, the edges `eo` and `ept` are both matched by the rule's LHS, where `eo` is found in the match $\mathbf{C}$ of the $\mathbf{C}^r$ part of the rule, and `ept` in $\mathbf{D}$ match of $\mathbf{D}^r$. Our objective is to construct the lifted transformation $\mathcal{R}_{EV}$ that produces M_2 when applied to M_1. We begin by constructing the graphical part of $\mathcal{R}_{EV}$ first, followed by the propositional part.

Graphical part. Consider applying R_{EV} to M_1 by directly applying it to M_b, M_1's base graph. Clearly, this approach does not produce the correct outcome. First, NAC_1 matches in M_b and thus the rule does not apply at all! Yet, there exists a concretization of M_1, m_{13}, for which neither of R_{EV}'s NACs match and thus R_{EV} should be applicable. We therefore expect it to be applicable to M_1 as well. Second, the RHS of the rule does not specify which elements in the output model should become Maybe, whereas M_2 clearly has them.

Thus, the classical strategy for rule application is not sufficient and needs to be augmented by the uncertainty in the model, i.e., the Maybe annotations of its elements. The presence of Maybe elements in the match of NAC_1 and in the match of the LHS of R_{EV} are both indications that R_{EV} applies to *some* concretizations but not others. We thus need to change the May model M_1 so that it represents both those concretizations that are unchanged by R_{EV} and those where the rule has been applied. Applying R_{EV} to a concretization of M_1 entails (1) deleting the edge `ept`, because it is included in the match $\mathbf{D}$ of $\mathbf{D}^r$, and (2) adding the elements of $\mathbf{A}$: `ge`, `gn`, `go`, `gt`, `gp`, and `epf`. However, we cannot altogether delete `ept` from M_1 because it should still remain in the unchanged concretization m_{11}. Instead, we must keep it annotated with Maybe to indicate that it is part of some concretizations but not others. Similarly, the newly added elements should be annotated with Maybe to indicate that they are added in m_{13} but not M_{11} or m_{12}.

We summarize the application of the graphical part of $\mathcal{R}_{EV}$ to M_1 as follows: (a) Apply R_{EV} to the base graph M_b of M_1 even though NAC_1 matches because the match contains a Maybe element. (b) Include both $\mathbf{D}$ and $\mathbf{A}$ in the base graph of M_2 and annotate all of their elements with Maybe because the match of the LHS in M_1 contains a Maybe element.

Propositional part. We now define the propositional part of $\mathcal{R}_{EV}$ that transforms the May formula Φ_1 of M_1 into Φ_2 of M_2. We achieve this by defining an operation on Φ_1 that has the effect of transforming the truth table of Φ_1 into the truth table of Φ_2. First note that the truth tables can be split into two parts:



Φ_1:

		$\overline{\mathbf{N}}$	C				D
		sn	e	eo	et	en	ept
Φ_{nchg}	m_{11}	1	1	1	1	1	1
	m_{12}	1	0	0	0	0	0
Φ_{chg}	m_{13}	0	1	1	1	1	1

$\overline{\mathbf{N}}$= {NAC_1}, LHS=$\mathbf{C}\cup\mathbf{D}$

Φ_2:

	$\overline{\mathbf{N}}$	C				D	A					
	sn	e	eo	et	en	ept	epf	ge	go	gt	gp	gn
m_{21}	1	1	1	1	1	1	0	0	0	0	0	0
m_{22}	1	0	0	0	0	0	0	0	0	0	0	0
m_{23}	0	1	1	1	1	0	1	1	1	1	1	1

legend: added unmodified modified

Fig. 6. Applying the lifted version of R_{EV} to the motivating example. Left: input model M_1, May graph and truth table of the May formula. Right: output model M_2, May graph and truth table of the May formula.

(a) The concretizations where R_{EV} does not apply (i.e., m_{11} and m_{12}). The corresponding rows m_{21} and m_{22} in Φ_2 remain unchanged, and the variables in $\mathbf{A}$ are set to False (denoted by 0) to indicate that $\mathbf{A}^r$ is not added. We denote the formula representing the unchanged part by Φ_{nchg}.

(b) The concretizations where R_{EV} does apply (i.e., m_{13}). The corresponding row m_{23} has the variables of $\mathbf{D}$ set to 0 to indicate that $\mathbf{D}^r$ is deleted and the variables of $\mathbf{A}$ set to 1 to indicate that $\mathbf{A}^r$ is added. We denote the formula representing the changed part by Φ_{chg}.

Thus, $\Phi_2 = \Phi_{nchg} \vee \Phi_{chg}$.

To obtain the unchanged part, we begin by specifying a condition, over elements of M_1, under which the rule R_{EV} applies, i.e., when its NAC $\mathbf{N}^r$ does not match in M_b and both $\mathbf{C}^r$ and $\mathbf{D}^r$ do match: $\Phi_{apply} = \neg\phi^{and}_{\mathbf{N}} \wedge \phi^{and}_{\mathbf{C}} \wedge \phi^{and}_{\mathbf{D}}$. Let $\phi^{and}_{\mathbf{X}}$ where $\mathbf{X} \in \{\mathbf{N},\mathbf{C},\mathbf{D}\}$ denote the conjunction of all variables in $\mathbf{X}$ that represent elements that are Maybe. Restricting Φ_1 to those concretizations of M_1 where R_{EV} does not apply ($\neg\Phi_{apply}$) and forcing the variables of $\mathbf{A}$ to become False produces the unchanged part: $\Phi_{nchg} = (\Phi_1 \wedge \neg\Phi_{apply}) \wedge \neg\phi^{or}_{\mathbf{A}}$, where $\phi^{or}_{\mathbf{A}}$ indicates the disjunction of all variables in $\mathbf{A}$ that represent elements that are Maybe.

For the changed part Φ_{chg}, we restrict Φ_1 to those concretizations of M_1 where R_{EV} does apply and force the variables of $\mathbf{D}$ to become False and those of $\mathbf{A}$ to become True: $\Phi_{chg} = (\Phi_1 \wedge \Phi_{apply})|^{\exists\mathbf{D}} \wedge \neg\phi^{or}_{\mathbf{D}} \wedge \phi^{and}_{\mathbf{A}}$. Here, $(\Phi_1 \wedge \Phi_{apply})|^{\exists\mathbf{D}}$ indicates *existential quantification* of all variables in $\mathbf{D}$ that occur in formula $\Phi_1 \wedge \Phi_{apply}$. In our example, $\mathbf{D} = \{\texttt{ept}\}$, so it becomes $(\Phi_1 \wedge \Phi_{apply})|_{\texttt{ept}=T} \vee (\Phi_1 \wedge \Phi_{apply})|_{\texttt{ept}=F}$. That is, we eliminate each variable in $\mathbf{D}$ from $\Phi_1 \wedge \Phi_{apply}$ by taking the disjunction of the cases where it is set to False and to True. Quantifying out variables in $\mathbf{D}$ is done before forcing them to become False (using $\neg\phi^{or}_{\mathbf{D}}$) because we are changing the values of existing variables (the variables of $\mathbf{D}$ already

occur in Φ_{apply}) and not just setting the value for new variables as we are for $\mathbf{A}$. Otherwise, we get an inconsistency because $\Phi_{apply} \Rightarrow \phi^{and}_{\mathbf{D}}$ by definition.

Substituting the variables from the example and simplifying gives:

$$\begin{aligned}\Phi_{nchg} =&(\texttt{sn} \wedge \texttt{e} \wedge \texttt{eo} \wedge \texttt{et} \wedge \texttt{en} \wedge \texttt{ept} \wedge \neg\texttt{epf} \wedge \neg\texttt{ge} \wedge \neg go \wedge \neg\texttt{gt} \wedge \neg\texttt{gt} \wedge \neg\texttt{gp} \wedge \neg\texttt{gn}) \vee \\ &(\texttt{sn} \wedge \neg\texttt{e} \wedge \neg\texttt{eo} \wedge \neg\texttt{et} \wedge \neg\texttt{en} \wedge \neg\texttt{ept} \wedge \neg\texttt{epf} \wedge \neg\texttt{ge} \wedge \neg\texttt{go} \wedge \neg\texttt{gt} \wedge \neg\texttt{gt} \wedge \neg\texttt{gp} \wedge \neg\texttt{gn}) \\ \Phi_{chg} =&(\neg\texttt{sn} \wedge \texttt{e} \wedge \texttt{eo} \wedge \texttt{et} \wedge \texttt{en} \wedge \neg\texttt{ept} \wedge \texttt{epf} \wedge \texttt{ge} \wedge \texttt{go} \wedge \texttt{gt} \wedge \texttt{gt} \wedge \texttt{gp} \wedge \texttt{gn})\end{aligned}$$

The resulting formula $\Phi_2 = \Phi_{nchg} \vee \Phi_{chg}$ is the same as the May formula in Fig. 3(d) and has the same truth table as the one shown in Fig. 6.

4.2 General case

We can generalize the above process to an arbitrary rule R and define how the graphical part of its lifted version $\mathcal{R}$ is applied to a May model M to produce a May model M'. As with the running example, we define this in terms of applying R to the base graph of M and then making modifications. Following Definition 4, the matching site for $\mathcal{R}$ is a matching site for R in the base graph of M.

Definition 8 (Lifted rule applicability conditions) *Given a May model M with a May formula Φ_M, a transformation rule $R = \langle\{NAC\}, LHS, RHS\rangle$, and a matching site $K = \langle\overline{\mathbf{N}}, \mathbf{C}, \mathbf{D}\rangle$, the lifted rule $\mathcal{R}$ is* applicable at K *iff the following conditions hold:*

1. *For all $N \in \overline{\mathbf{N}}$, N contains a* Maybe *element*
2. *$\Phi_M \wedge \Phi_{apply}$ is satisfiable, where $\Phi_{apply} = \neg \bigvee\{\phi_N | N \in \overline{\mathbf{N}}\} \wedge \phi^{and}_{\mathbf{C}} \wedge \phi^{and}_{\mathbf{D}}$.*

In this definition, Condition 1 ensures that there is no NAC match without Maybe elements; otherwise the NAC match would necessarily occur in every concretization and so R would not apply to *any* concretization of M. Condition 2 uses the constraints in the May formula to ensure that at this matching site, the rule R matches in at least one concretization of M. Specifically, this checks that there exists a concretization in which all f the Maybe elements of $\mathbf{C}$ and $\mathbf{D}$ are True and not all of the Maybe elements in any NAC match are set to True.

We now give the general definition of a rule application for a lifted rule.

Definition 9 (Lifted rule application) *Given a May model M with a May formula Φ_M, a transformation rule $R = \langle\{NAC\}, LHS, RHS\rangle$ and matching site $K = \langle\overline{\mathbf{N}}, \mathbf{C}, \mathbf{D}\rangle$ in M for which the rule applicability conditions are satisfied,* the lifted rule $\mathcal{R}$ is applied *to produce a May model M' as follows:*

1. *if K contains no* Maybe *elements, apply R in the classical way to produce the base graph of M' and set $\phi_{M'} = \phi_M$.*
2. *otherwise,*
   (a) *set $M' = M$;*
   (b) *add the elements $\mathbf{A}$ of the $\mathbf{A}^r$ part of the RHS to M';*
   (c) *annotate all elements of $\mathbf{A}$ and $\mathbf{D}$ with* Maybe*;*
   (d) *set $\Phi_{M'} = [(\Phi_M \wedge \neg\Phi_{apply}) \wedge \neg\phi^{or}_{\mathbf{A}}] \vee [(\Phi_M \wedge \Phi_{apply})|^{\exists \mathbf{D}} \wedge \neg\phi^{or}_{\mathbf{D}} \wedge \phi^{and}_{\mathbf{A}}]$.*

In this definition, Case 1 captures the situation when there are no Maybe elements at the matching site and so the rule can be applied in the classical way and the May model is unaffected. Case 2 captures the situation when there are Maybe elements in parts of the matching site so that R may apply in some concretizations but not in others. This case mirrors the discussion of $\mathcal{R}_{EV}$ in Section 4.1. In particular, in the graphical part, $\mathbf{D}^r$ is not deleted (step a) but $\mathbf{A}^r$ is still added (step b) and all of the elements in $\mathbf{A}$ and $\mathbf{D}$ are set to Maybe (step c). The propositional part (step d) is the same as for the $\mathcal{R}_{EV}$ example except that Φ_{nchg} and Φ_{chg} are inlined and the more general case of $\overline{\mathbf{N}}$ is used in Φ_{apply} (from Definition 8) to account for multiple NAC matches that could exist in the base graph of M.

As with a classical rule system, lifted rules continue to be applied until no rule is applicable. Note that the resulting model M' may not necessarily be in GRF after every rule application. That is, M' can contain redundant Maybe elements. If M' is intended for human consumption (as opposed to automated reasoning) then the additional step of putting it into GRF is advisable. However, this step is optional since it does not affect the set of concretizations that the May model represents.

4.3 Analysis

In this section, we discuss some key properties of lifted rules such as their correctness, termination and confluence. The resulting properties apply to *arbitrary* transformations being lifted, whether they are injective, endogenous, exogenous and so on.

Correctness. We now show that lifting described by Definitions 8 and 9 satisfies the correctness condition in Definition 7. Specifically, we show that if a lifted rule $\mathcal{R}$ is applied to a May model M to produce a May model M', then the concretizations of M' must be exactly the set obtained by applying the classical rule R to each concretization of M. We focus our argument on a specific matching site $K = \langle \overline{\mathbf{N}}, \mathbf{C}, \mathbf{D} \rangle$ since by transitivity, if the rule is correct when applied to each site, then the application to any sequence of sites is also correct.

We begin with checking correctness of the applicability condition (Definition 8) of the lifted rule $\mathcal{R}$: whenever R is applicable for some concretization of M at K, then $\mathcal{R}$ is also applicable, i.e., $\mathcal{R}$ it does not miss any sites where a concretization can be affected by R. By Condition 1 of Definition 8, if there is a NAC in $\overline{\mathbf{N}}$ that has no Maybe elements then $\mathcal{R}$ does not apply at K. But a NAC without Maybe in the base graph of M means that this NAC appears in every concretization of M and thus the classical rule R does not apply to any concretization either and thus applying the lifted rule does not miss any classical rule applications.

Condition 2 says that $\Phi_M \wedge \Phi_{apply}$ must be satisfiable for $\mathcal{R}$ to apply, which happens iff there exists a concretization of M where $\mathbf{C}^r$ and $\mathbf{D}^r$ are present and no NAC in $\overline{\mathbf{N}}$ is present – exactly the classical applicability condition in Defini-

tion 5. If this condition does not hold, there are no classical rule applications in any concretization, therefore the lifted rule applicability condition is correct.

We now argue that the lifted rule application in Definition 9 is correct. To do this, we show that if $\mathcal{R}$ satisfies the applicability conditions, then applying $\mathcal{R}$ at a site K has the same effect as applying R at K in each concretization. Case 1 says that when K contains no Maybe elements, we apply the rule classically to the base graph of M. Without Maybe elements, K occurs in every concretization of M and so the classical application of R in every concretization would be identical to applying $\mathcal{R}$.

Case 2 applies when K has some Maybe elements. In this case, the concretizations are split into those where R does not apply and those where it does. We then aim to show that the steps (a-d) for constructing the graphical and propositional effect of applying $\mathcal{R}$ are "correct by construction". We do not repeat this argument, described in Sec. 4.1, here, for brevity. Thus, we conclude that the lifted rule application is also correct. Since both the applicability condition and the effect of application are correct, we conclude that $\mathcal{R}$ satisfies the specification of correct lifting in Definition 7.

Termination. To prove termination, we show that if an application of a set of classical rules on an input model always terminates than so does the set of the corresponding lifted rules. Without loss of generality, we restrict ourselves to a rule set containing a single classical rule R which we assume is terminating. Since $\mathcal{R}$ is correct according to Definition 7, repeatedly applying it to a May model M has the same effect as repeatedly applying R to each concretization of M. Since R is terminating, it eventually is no longer applicable to any concretization of M. At this point, Φ_{apply} which encodes classical applicability is False and thus $\Phi_M \wedge \Phi_{apply}$ is not satisfiable, and, by Condition 2 of Definition 8, $\mathcal{R}$ does not apply. Thus, when the application of R terminates, the application of $\mathcal{R}$ terminates as well. Therefore, if R is terminating, so is $\mathcal{R}$.

Confluence. We argue that if a set of classical rules is confluent then the corresponding set of lifted rules is also confluent "up to an equivalence", that is, when the process terminates, the resulting May model has the same set of concretizations, regardless of the order in which the rules have been applied. Repeatedly applying lifted rules to a May model M has the same effect as repeatedly applying the corresponding classical rules to each concretization of M. Since the classical rules are confluent and terminating, the process over lifted rules reaches the same final set of concretizations. Thus, the lifted rule set is confluent "up to an equivalence".

5 Evaluation

We applied our lifting approach to the problem of mapping simple UML class diagrams to relational database schemas. This problem is called "Object-Relational Mapping" (ORM) and is often used as a benchmark for model transformations [2]. Our aim was to gather evidence about how the lifting approach scales as

Table 1. Results of applying the ORM rules to the Ecore metamodel.

Number of concretizations:	1	24	48	108	144	192	256
Number of Maybe elements:	0	5	6	8	10	12	14
Time (sec):	32.6	32.8	32.7	32.9	32.6	33.0	48.4
Size of May formula (KiB):	0	27.9	14.0	1,080.9	1,153.4	19,361.9	320,570.7

uncertainty increases. We thus measured the runtime of performing ORM with lifted rules while increasing levels of uncertainty and compared it with the baseline runtime of performing ORM for a classical model. The ORM transformation rules we used came from [26] and consist of 5 layered transformation rules that, given a class diagram, create a relational schema and traceability links.

We used the class diagram specification of the Ecore metamodel [24] as input to the ORM rules. Serializing Ecore models in a database is an important technical problem that has resulted in the establishment of two Eclipse projects, *CDO* [7] and *Teneo* [8], both of which implement ORM for Ecore. We manually flattened the Ecore metamodel and adapted it to the type graph used by the ORM rules in [26]. The resulting model consisted of 65 model elements: 17 classes, 17 associations, 6 generalization links and 25 attributes. Starting with a May model with a single concretization (no uncertainty), we gradually increased the degree of uncertainty by adding more concretizations, by a step of roughly 50, thus creating models with 1, 24, 48, 108, 144, 192, and 256 concretizations. To accomplish that we incrementally injected points of uncertainty, annotating elements with Maybe and creating the corresponding May formulas. The most uncertain case (256 concretizations) contained 8 points of uncertainty, expressed across a total of 14 Maybe elements.

We implemented the lifting of the ORM rules using Henshin [1]. For the satisfiability check required in Definition 8, we used the Z3 SMT solver [5]. We used the Model Management Tool Framework [17] as the integration platform. We executed the case study on a computer with Intel Core i7-2600 3.40GHz×4 cores (8 logical) and 8GB RAM, running Ubuntu-64 12.10. We applied the set of lifted rules to each input May model and recorded the total runtime and the size of the resulting May formula. Our observations are shown in Table 1.

The results show that the total runtime remains almost constant at roughly 32.8 seconds, except for the largest category where it increases to 48.4 seconds. On the other hand, we see a dramatic increase in the size of the May formula, from 27.9 KiB for the smallest category, to approx. 320.6 MiB for the largest. This exponential growth in size is reasonable, given (Definition 9). Overall, the results suggest that lifting scales reasonably with respect to time, whereas the increasing size of the May formula may be a problem. However, we note that our implementation did not attempt to incorporate any formula simplification heuristics, and therefore there is room for optimization.

6 Related Work

The notion of uncertainty addressed by May models captures the scenario of having multiple possible alternative design solutions, with the modeler being unsure about which one to pick. Discussion of work related to representing sets of models is out of scope of the current paper; a thorough comparison of May models with related formalisms can be found in [12]. May models encode a set of classical models, and lifted rules are rules that can transform entire sets of models simultaneously. In the following, we discuss work related to transformations that apply to modeling formalisms that represent sets of models.

Different variants of feature models have been proposed in the literature to encode a set of possible configurations of a software product line [22]. Transformations of feature models, i.e., the creation of a feature model representing a subset of the original, have been studied in [4] under the name of *feature model specialization*. This process is described as a series of operations such as "feature cloning" and "reference unfolding" and resembles the uncertainty-reducing transformation of [10]. Graph transformations have also been applied to feature models, e.g., in [23], they are used to refactor product lines via feature model merging. Transformations that apply to metamodel definitions also transform sets of models, i.e., the set of possible instances of the metamodel. The Object-to-Relational Mapping transformation [26] in Sec. 6 is one such example. Similarly, special purpose transformation languages have been built to transform ontologies, such as a rule based language based on xOWL [27].

The main difference between these transformations and the lifting approach presented here is that they are tailored to specific tasks, whereas lifting applies to *arbitrary* transformation rules. Moreover, these techniques only indirectly affect the classical models (e.g., variants or instances) represented by the abstraction formalism. On the other hand, lifted transformations match and transform the alternatives directly, via the propositional part of the lifted rules.

7 Conclusion

In this paper, we have shown how to adapt existing model transformations to May models, a formalism that allows uncertainty to be explicated in software artifacts. To achieve this, we have introduced the process of *lifting* graph transformation rules and proved its correctness of application to May models. We have implemented our approach and applied it to the Object-Relational Mapping benchmark. Our experience showed that the overhead of applying lifted transformations is reasonable, so we feel that the approach is feasible for transforming realistic models with uncertainty. In the future, we are planning to implement lifting as a higher-order transformation (HOT). We expect the approach to lift a classical rule to a layered graph grammar of classical rules, allowing us to implement a dedicated tool by reusing existing graph grammar implementations such as Henshin [1]. We also intend to evaluate our approach further and expand our lifting technique to models containing other types of uncertainty [20].

References

1. T. Arendt, E. Biermann, S. Jurack, C. Krause, and G Taentzer. "Henshin: advanced concepts and tools for in-place EMF model transformations". In *Proc. of MODELS'10*, pages 121–135, 2010.
2. J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt. "Proceedings of the Model Transformations in Practice Workshop". In *Satellite Events at the MoDELS 2005 Conference*, 2006.
3. E. Casais. "The Automatic Reorganization of Object Oriented Hierarchies – A Case Study". *Object Oriented Systems*, 1:95–115, 1994.
4. K. Czarnecki, S. Helsen, and U. Eisenecher. "Staged Configuration Using Feature Models". In *Proc. of SPLC'04*, pages 266–283, 2004.
5. L. De Moura and N. Bjørner. "Satisfiability Modulo Theories: Introduction and Applications". *Commun. ACM*, 54(9):69–77, September 2011.
6. C. Ebert and J. De Man. "Requirements Uncertainty: Influencing Factors and Concrete Improvements". In *Proc. of ICSE'05*, pages 553–560, 2005.
7. Eclipse. *CDO website: http://www.eclipse.org/cdo/*, accessed 2013-03-16.
8. Eclipse. *Teneo website: http://wiki.eclipse.org/Teneo/*, accessed 2013-03-16.
9. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, 1 edition, 2006.
10. M. Famelis, M. Chechik, and R. Salay. "Partial Models: Towards Modeling and Reasoning with Uncertainty". In *Proc. of ICSE'12*, 2012.
11. M. Famelis, M. Chechik, and R. Salay. "The Semantics of Partial Model Transformations". In *Proc. of MiSE'12*, 2012.
12. M. Famelis, M. Chechik, and R. Salay. "Towards Modeling and Reasoning with Uncertainty", 2013. Submitted.
13. H. Ibrahim, B. H. Far, A. Eberlein, and Y. Daradkeh. "Uncertainty Management in Software Engineering: Past, Present, and Future". In *Proc. of CCECE'09*, pages 7–12, 2009.
14. M. Famelis and S. Santosa. "MAV-Vis: a Notation for Model Uncertainty". In *Proc. of MiSE'13*, 2013.
15. T. Mens, N. Van Eetvelde, S. Demeyer, and D Janssens. "Formalizing Refactorings with Graph Transformations". *Journal of Software Maintenance and Evolution: Research and Practice*, 17(4):247–276, 2005.
16. M. Sabetzadeh, S. Nejati, M. Chechik, and S. Easterbrook. "Reasoning about Consistency in Model Merging". In *Proc. LWI'10*, 2010.
17. R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh, and P. Viriyakattiyaporn. "An Eclipse-Based Tool Framework for Software Model Management". In *Proc. of Eclipse'07*, pages 55–59, 2007.
18. R. Salay, M. Chechik, M. Famelis, and J. Gorzny. "Verification of Uncertainty Reducing Model Transformations", 2013. Submitted.
19. R. Salay, M. Chechik, and J. Gorzny. "Towards a Methodology for Verifying Partial Model Refinements". In *Proc. of VOLT'12*, 2012.
20. R. Salay, M. Famelis, and M. Chechik. "Language Independent Refinement using Partial Modeling". In *Proc. of FASE'12*, 2012.
21. P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. "Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems". In *Proc. of RE'10*, pages 95–103, 2010.

22. P.Y. Schobbens, P. Heymans, and J.C. Trigaux. “Feature diagrams: A survey and a formal semantics”. In *Proc. of RE'06*, pages 139–148, 2006.
23. S. Segura, D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Generative and Transformational Techniques in Software Engineering II. chapter Automated Merging of Feature Models Using Graph Transformations, pages 489–505. 2008.
24. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2. edition, 2009.
25. A. van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.
26. D. Varró, S. Varró-Gyapay, H. Ehrig, U. Prange, and G. Taentzer. “Termination Analysis of Model Transformations by Petri Nets”. In *Proc. of ICGT'06*, pages 260–274, 2006.
27. L. Wouters and M.P. Gervais. “Ontology Transformations”. In *Proc of EDOC'12*, pages 71–80, 2012.
28. H. Ziv, D.J. Richardson, and R. Klösch. “The Uncertainty Principle in Software Engineering”, 1996. unpublished.