Supporting variability exploration and resolution during model migration

Davide Di Ruscio[1], Juergen Etzlstorfer[2], Ludovico Iovino[3], Alfonso Pierantonio[1], and Wieland Schwinger[2]

[1] Department of Information Engineering, Computer Science and Mathematics
Università degli Studi dell'Aquila - L'Aquila (Italy)
`name.surname@univaq.it`

[2] Department of Cooperative Information Systems
Johannes Kepler University Linz - Linz (Austria)
`name.surname@jku.at`

[3] Gran Sasso Science Institute - L'Aquila (Italy)
`ludovico.iovino@gssi.infn.it`

Abstract. In Model-Driven Engineering (MDE) metamodels are pivotal entities that underpin the definition of models. Similarly to any software artifact, metamodels evolve over time due to evolutionary pressure. However, whenever a metamodel is modified, related models may become invalid and adaptations are required to restore their validity. Generally, when adapting a model in response to metamodel changes, more than one migration strategy is possible. Unfortunately, inspecting all of them, which greatly overlap one with another, can be prone to errors. In this paper, we present an approach supporting the identification of variability during model migration and selection of migration alternatives by generating an *intensional* and thus concise representation of all migration alternatives by including also an explicit visualization of conflicting solutions.

1 Introduction

In Model-Driven Engineering [24] (MDE) metamodels are often considered a pivotal concept used for formalizing and describing application domains. A wide range of artifacts, tools and applications are defined upon one or more metamodels that altogether form a modeling ecosystem [6]. Generic modeling platforms (e.g., ADOxx[4], EMF[5], and Metaedit[6]) enable the development of full-fledged modeling environments that are specifically tailored around organization needs [8,14]. Similarly to any other software artifact, metamodels are prone to evolution during their routinely use, to cope with improvements, extensions, and corrections [18]. However, any change to a metamodel can endanger the integrity and consistency of the modeling ecosystem as models, transformations, or

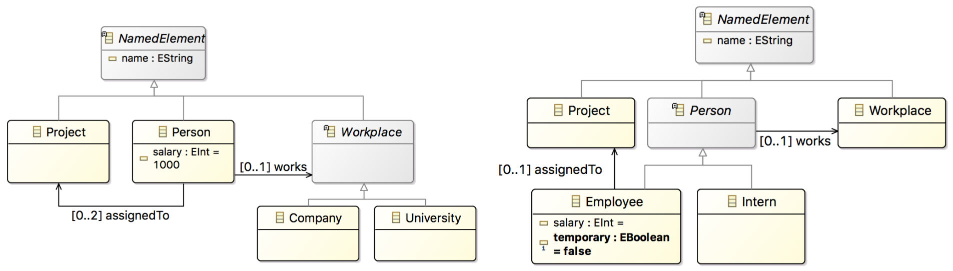
[4] http://www.adoxx.org
[5] http://eclipse.org/modeling/emf/
[6] http://www.metacase.com/products.html

even editors might become invalid [7]. The metamodel co-evolution (or coupled evolution) problem concerns the process of recovering the relationship between evolving metamodels and the dependent artifacts in the modeling ecosystem [7]. In this paper, we focus on the *model co-evolution* problem, i.e., on the process of migrating a model to restore the conformance relation between evolving metamodels and those models affected by the metamodel changes.

Over the last decade, numerous approaches for co-evolution of metamodels and models have been proposed. Most of them can be distinguished by falling into the groups of *inductive* and *prescriptive* ones: the former ones (e.g., [4,12]) automatically derive a model migration procedure from the metamodel differences, while in the course of the latter ones models are programmatically migrated by means of predefined procedures (e.g., [13,21,29]). An aspect that has been largely neglected so far is the following: when migrating a model in response to a metamodel change there might be multiple alternatives to restore its conformance. For instance, if the multiplicity of an association in a metamodel is decreased, there are many ways of selecting the exceeding associations to be removed from the instance models. Identifying the right migration alternative is a challenging task as it should consider also aspects that go beyond the mere conformance recovering, such as information erosion [25] and reducing the number of model changes. Recently, an approach has been proposed to mitigate such difficulties by generating *all* possible migrations at once [25]. Then, the responsibility of identifying the *right* model migration is shifted from the implementer of the migration program to the modeler, who can then inspect the solution space and identify the most adequate solution. Unfortunately, already little changes in the metamodel usually give place to a multitude of possible model migrations that are difficult to inspect as they greatly overlap one with another.

In this paper, we present an approach to alleviate the consequences of dealing with the multitude of model migrations that can restore model conformance. The purpose of the approach is to help the modeler in finding the co-evolution for models by supporting the modeler with a proper visualization of potential conflicting solutions. Instead of *extensionally* [20] generating all migrated model as done in [25], an *intensional* representation of them is given. In essence, the approach permits to represent different solutions as a model with *variability* that indicates which parts of the solution are different for each migration alternative and is able to indicate if there are conflicts between solutions. The overall solution space is represented by a feature model [1] to better navigate alternatives and identify the wanted migration alternative. In addition, traceability between the individual metamodel changes and the corresponding migration alternatives is also provided in order to record modeler decisions and avoid to deal with already resolved variability.

Outline. In Sect. 2 a motivating example is given to illustrate how migration strategies can proliferate. Sect. 3 introduces the approach by presenting the variability metamodel for the intensional representation of the different solutions and illustrates it on the motivating example. The approach is critically discussed in



(a) initial version (b) evolved version

Fig. 1: The Simple Workplace Metamodel (SWMM)

Sect. 4. In Sect. 5 related work is considered and, Sect. 6 draws some conclusions and outlines future plans.

2 Motivating Example

In order to satisfy unforeseen requirements or to better represent the considered application domain, metamodels can be subject to modifications as for instance in the case of the Simple Workplace MetaModel (SWMM) shown in Fig. 1.a[7]. In particular, let us suppose that a number of changes have been performed on the SWMM metamodel leading to the evolved version shown in Fig. 1.b. More specifically, the performed changes (or refactorings) shall be the following:

R1. Introduce subclasses: the metaclasses Employee and Intern have been added as subtype of Person that becomes abstract.

R2. Push down attribute: the attribute salary has been pushed down in the hierarchy, from Person to Employee.

R3. Add mandatory attribute: the mandatory attribute temporary has been added to the metaclass Employee.

R4. Restrict reference cardinality: the multiplicity of reference assignedTo has been restricted from [0..2] to [0..1].

R5. Flatten hierarchy: the metaclasses Company and University have been removed, flattening the hierarchy of Workplace.

A simple workplace model conforming to the initial version of SWMM is shown in Fig. 2. The model specifies an instance of the metaclass Person named John: he works at the University of L'Aquila and is employed in two projects, namely LearnPad and MDEForge. Such a model is no longer conforming to the newer version of the SWMM metamodel, therefore it has to be migrated in order to re-establish the lost conformance relationship. In particular, the following elements violate the conformance relationship:

[7] For the sake of clarity, abstract classes are depicted in gray.

- John:Person and Adele:Person cannot be instances of the metaclass Person, which is now abstract; in addition, such instances contain the reference assignedTo and the attribute salary that have been removed from the Person metaclass;
- the Univaq:University element cannot be in the model because the metaclass University has been flattened into Workplace;
- the number of assigned projects to John:Person is higher than 1 which is the new maximal number of projects that can be assigned to Person.

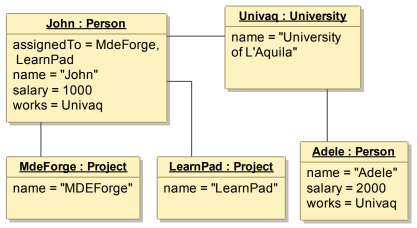


Fig. 2: A simple workplace model

In general, various migration procedures to recover the conformance are possible, each providing a different solution. Thus, it is of utmost importance to inspect the different alternatives for detecting the one, which fits modeler's needs best. However, because the alternatives largely overlap each other and might present conflicts among them, the procedure can be tedious and prone to errors if executed without (semi) automated support. For instance, because of the SWMM metamodel refactoring the simple workplace model in Fig. 2 can be migrated by means of several *model migrations* as reported in Table 1 and explained in the following[8]:

R1. Introduce subclasses: this metamodel change involving the metaclass Person can be resolved by means of any of the following alternatives:

- *R1a1:* all instances of Person are removed;
- *R1a2, R1a3:* all instances of the abstract superclass Person are re-typed into either Employee (*R1a2*) or Intern (*R1a3*);
- *R1a4:* a non-empty set of instances is re-typed to Employee while another non-empty set of instances is re-typed to Intern; the decision criteria about which instances are retyped to one or the other type has to be provided by the user in form of, e.g., OCL expressions.

R2. Push down attribute: this change, which pushed down the attribute salary from Person to Employee, can be resolved by operating one of the following model migrations:

- *R2a1:* retain the value of the pushed attribute;
- *R2a2:* delete the value of the pushed attribute;

R3. Add mandatory attribute: the addition of the mandatory attribute temporary can be resolved by setting its value either to *true* (*R3a1*), or to *false* (*R3a2*). This should be decided by the user.

[8] Please note that each migration alternative is identified by a term like *R1a1* where *a1* is one of the possible migration alternative related to the metamodel change *R1*

Table 1: Possible model migration alternatives for the motivating example

Metamodel change	**Possible migration alternatives**
R1. Introduce subclasses	*R1a1.* Remove the existing instances of type Person
	R1a2. Re-type the existing instances from Person to Employee
	R1a3. Re-type the existing instances from Person to Intern
	R1a4. Re-type the existing instances from Person to Employee or Intern with different (non-empty) combinations
R2. Push down attribute	*R2a1.* Maintain the attribute value of salary in the re-typed instance
	R2a2. Remove the attribute value of salary
R3. Add mandatory attribute	*R3a1.* Set the attribute value of Employee.temporary to true
	R3a2. Set the attribute value of Employee.temporary to false
R4. Restrict reference cardinality	*R4a1.* Remove one of link to the project assigned to a Person[9]
	R4a2. Remove all the links of project related to a Person
	R4a3. Re-assign one of the project to other persons[9]
	R4a4. Re-assign all the project to other persons[9]
R5. Flatten hierarchy	*R5a1.* Re-type all the instance with the corresponding flattened subclasses with the supertype
	R5a2. Remove all instances of Workplace

R4. Restrict reference cardinality: this change operated on the reference assignedTo can be resolved applying one of the following migration alternatives:
- *R4a1:* unassign one of the two Projects from a Person;
- *R4a2:* unassign all Projects to allow for a complete reassignment;
- *R4a3, R4a4:* reassign one Project instance (R4a3) or all instances of Project (R4a4) to another instance of Person.

R5. Flatten hierarchy: this modification, which affected Workplace, Company, and University, can be resolved by re-typing all the instances of University or Company to the superclass Workplace (*R5a1*) or by deleting all of them (*R5a2*).

A *migration solution* consists of a combination of selected migration alternatives, one for each metamodel refactoring, which are not in conflict to each other. However, alternatives can be combined in different manners by exponentially increasing the number of migration solutions and thus the complexity of the problem. For instance, by considering the 5 changes operated on the initial SWMM metamodel of the previous example, the total number of possible migration solutions for the sample workplace model are 128 ($= 4 \times 2 \times 2 \times 4 \times 2$), although this might be an over-approximation because conflicts might occur between migration alternatives as discussed later on the paper. However, if user-specified decision criteria are allowed, the number might be even higher.

[9] The selection criteria can be decided by the generation process, e.g. first, last, random

3 Approach

In this section, we present an approach to represent, explore, and select migration alternatives in response to metamodel changes. The approach allows to *intentionally* represent multiple solutions for the model migration problem at hand. In particular, instead of *extensionally* represent all the possible solutions as typically done by existing techniques (e.g., [25]), a single model with *variability* is employed to precisely denote which parts of the solution are different for each migration alternative. The proposed approach also permits to highlight aspects that are not evident with classical approaches, such as conflicting alternatives.

Figure 3 shows the main artifacts and activities of the proposed approach. The main concepts are represented by the Variability Weaving Metamodel (WMM), which employes model weaving [2] (see m_{WMM}) by linking different models to represent all possible migration solutions that can be alternatively applied on the initial model m_{MM} in order to obtain models conforming to the evolved metamodel (MM'). In particular, for each metamodel change the weaving model represents corresponding migration alternatives for m_{MM}. Some of those alternatives might be in conflict with others, e.g., the deletion of an element is in conflict with other operations consuming it. In order to make the visualization of alternatives and their conflicts easier to be analyzed, a model transformation is applied on the source model m_{WMM} to generate a target feature model [1]. The generated feature model can be inspected by the user in order to chose a valid combination of migration alternatives, to finally obtain a model m' conforming to MM'.

In the remaining of the section all the parts of the approach shown in Fig. 3 are described.

3.1 Variability metamodel for representing different migration solutions

The variability metamodel WMM previously mentioned is shown in Fig. 4 and has been constructed by building upon our previous work on difference representation for metamodels [3], but shifting the concepts from the M2 level of

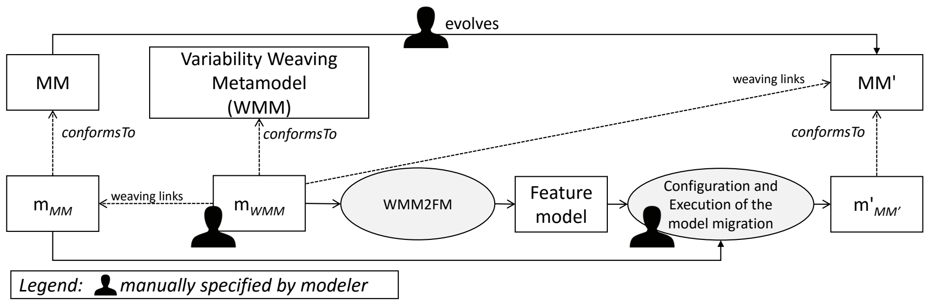

evolves
MM
Variability Weaving Metamodel (WMM)
weaving links
MM'
conformsTo
conformsTo
conformsTo
m_MM
weaving links
m_WMM
WMM2FM
Feature model
Configuration and Execution of the model migration
m'_MM'
Legend: manually specified by modeler

Fig. 3: Proposed Approach

the OMG modeling stack to the M1 level [16]. Since we employ the Eclipse technology stack, Ecore serves as meta-metamodel of the proposed variability metamodel.

The metamodel consists of the root metaclass VariabilityModel that serves as a container for all migration Solutions, each of which is performed on the affected model to restore its conformance with respect to the newer metamodel version. In order to express all the possible migration strategies, each Solution consists of one or more disjunct Alternatives. Each alternative (as those illustrated in Table 1) is represented in terms of effects on the model to be migrated. To this end, the DiffInstance and DiffFeature metaclasses have been introduced: the former identifies the model element affected by the metamodel refactoring, whereas the latter identifies the corresponding structural features. The metamodel is capable of describing all added, deleted, and changed metamodel instances along with their added, deleted, and changed features. Additionally, instances that remain the same, i.e., migration is not needed, can be specified (cf. CopyInstance). Thus, all possible migration alternatives can be represented. Please note that applicationElement is the reference to an element subject to change in m. The properties name and featureName, value, newValue represent the changed/new values in changed/new instances and features, respectively.

As already mentioned above, migration alternatives might also be in conflict with each other. For example, considering a deletion of an element as a possible alternative, this alternative is in conflict with all other alternatives that still depend on the existence of this element. As shown in Fig. 4, an Alternative might be in conflict with more than one other migration alternatives (see the reference conflictsWith). In the following, we present a way how users can deal

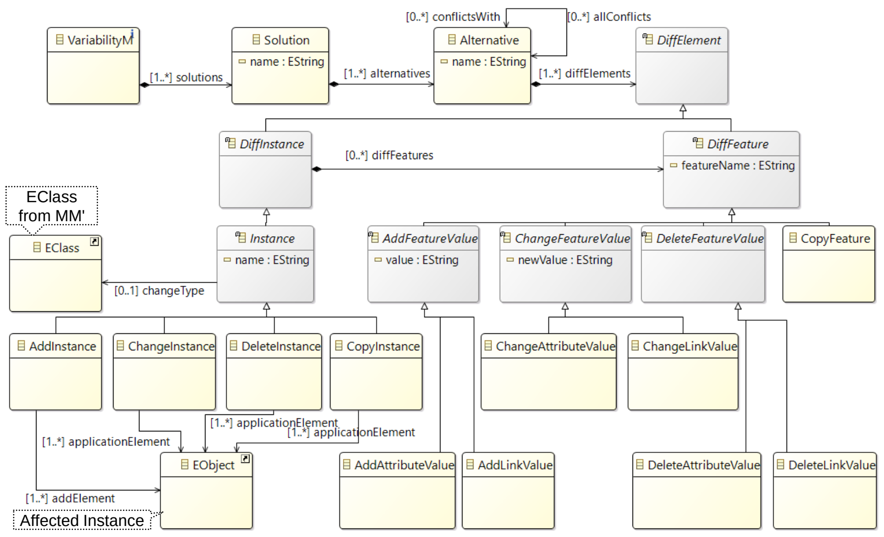
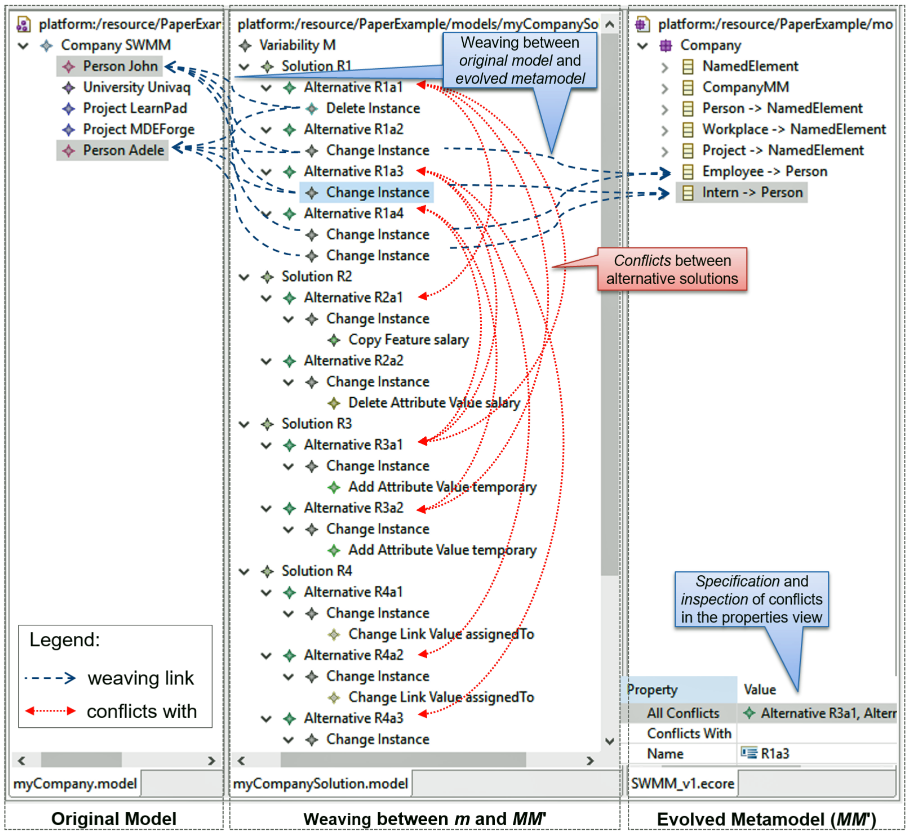


Fig. 4: Variability Weaving Metamodel (WMM)



```
platform:/resource/PaperExar
  Company SWMM
    Person John
    University Univaq
    Project LearnPad
    Project MDEForge
    Person Adele
```

```
platform:/resource/PaperExample/models/myCompanySo
  Variability M
    Solution R1
      Alternative R1a1
        Delete Instance
      Alternative R1a2
        Change Instance
      Alternative R1a3
        Change Instance
      Alternative R1a4
        Change Instance
        Change Instance
    Solution R2
      Alternative R2a1
        Change Instance
          Copy Feature salary
      Alternative R2a2
        Change Instance
          Delete Attribute Value salary
    Solution R3
      Alternative R3a1
        Change Instance
          Add Attribute Value temporary
      Alternative R3a2
        Change Instance
          Add Attribute Value temporary
    Solution R4
      Alternative R4a1
        Change Instance
          Change Link Value assignedTo
      Alternative R4a2
        Change Instance
          Change Link Value assignedTo
      Alternative R4a3
        Change Instance
```

```
platform:/resource/PaperExample/mo
  Company
    NamedElement
    CompanyMM
    Person -> NamedElement
    Workplace -> NamedElement
    Project -> NamedElement
    Employee -> Person
    Intern -> Person
```

Property	Value
All Conflicts	Alternative R3a1, Alterr
Conflicts With	
Name	R1a3

Weaving between *original model* and *evolved metamodel*
Conflicts between alternative solutions
Specification and *inspection* of conflicts in the properties view
Legend:
weaving link
conflicts with
myCompany.model
myCompanySolution.model
SWMM_v1.ecore
Original Model
Weaving between *m* and *MM'*
Evolved Metamodel (*MM'*)

Fig. 5: Sample weaving model represented by means of Epsilon ModeLink

with conflicts when selecting migration alternatives. This enables an explicit management of conflicts in subsequent stages of the migration process.

3.2 Variability model as weaving model

Employing the approach to our example, all migration alternatives shown in Table 1 have been represented by means of the weaving model shown in Fig. 5 and conforming to the variability metamodel in Fig. 4. The model m_{WMM} has been manually specified by the user by exploiting the Eclipse Epsilon's model weaving facilities *ModeLink*[10].

On the left-hand side of Fig. 5 the sample workplace model conforming to the initial version of SWMM is shown, whereas the right-hand side of the figure shows the evolved version of SWMM. In the middle, all the weaving elements representing the possible migration alternatives of the sample workplace model are

[10] https://www.eclipse.org/epsilon/doc/modelink/

shown. In particular, the weaving model consists of links (annotated by dashed lines) relating model elements that have to be migrated (see the left-hand side of the figure), with metaclasses in the newer metamodel (see the right-hand side of the figure). Weaving links are organized in solutions, each consisting of migration alternatives. For instance, the solution for the metamodel change *R1 - Introduce subclasses* applied to the metaclass Person consists of four alternatives (R1a1 – R1a4), each representing the corresponding model migration. In particular,

– the alternative R1a1 contains a Delete Instance that refers to John and Adele, meaning that this choice deletes both instances.
– the alternative R1a3 links the instances John and Adele to the class Intern of the new SWMM via a Changed Instance element, meaning that they are re-typed to be instances of Intern during migration.

Since not all migration alternatives might be compatible with each other, WMM also allows to specify conflicts to declare disjunct alternatives. For instance, the property view on the lower right-hand side of Fig. 5 shows the specified conflicts for R1a3, which is in conflict with the alternative R3a1. In particular, R1a3 retypes all the instances of Person to Interns, whereas in R3a1 the attribute temporary, which is not existing in the Intern, is set to true. Conflicts are annotated in Fig. 5 by means of (vertical) dotted lines connecting the weaving model elements.

Please note that for the sake of clarity not all weaving links and conflicts are shown in Fig. 5, nevertheless all weaving links and conflicts regarding the R1 alternatives are shown.

3.3 Variability model as feature model

Feature models [1] are a compact representation of different configurations for a system, e.g., software product lines. In our approach, we employ feature models to provide a suitable representation of all Solutions along with their migration Alternatives, to support the user in identifying the right migration alternative. Therefore, Solutions are represented as mandatory features (since for each change a solution has to be chosen), while migration Alternatives are disjunct subfeatures of Solutions, thus the user can only decide for one concrete alternative at a time. However, since alternatives might be in conflict among them, we exploit constraints as part of the feature model to define these conflicts.

For instance, in order to provide a convenient representation for the solution and alternatives illustrated in Fig. 5, the feature model in Fig. 6 can be used. It can be automatically generated from the weaving model using the model transformation shown in Listing 1.1. The generated feature model consists of mandatory elements R1–R5 representing the different solutions that have to be considered to migrate the simple workplace model. Each of them consists of several disjunct alternatives that represent the concrete migration actions to be undertaken. If conflicts have been specified, they are denoted in the feature model by means of constraints, e.g., by $R1a1 \Rightarrow \neg R2a1 \wedge \neg R3a1 \wedge \neg R4a2$, thus,

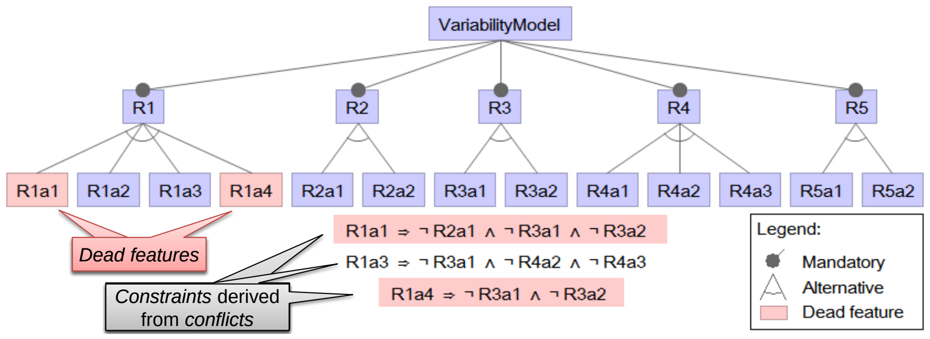


Fig. 6: Feature model related to the running example

excluding specific alternative combinations. Interestingly, the presence of constraints in mandatory alternatives might reveal "dead features", which means that in order to apply all solutions, some choices might not be valid. In the example, R1a1 and R1a4 are detected as dead features, since they either delete the instance John or retype it to Intern, thereby hindering all subsequent migration actions that rely on the instance itself or features of Employee. Besides dead features, all other conflicts lead to constraints that might restrict some solutions. For example, the choice of R1a3 will prevent the modeler from choosing also R3a1, R4a2, and R4a3.

3.4 The *WMM2FM* transformation

Feature models like the one in Fig. 6 can be automatically generated from a weaving model by means of a model transformation. Listing 1.1 provides such a transformation, which has been developed with the Epsilon Generation Language (EGL) [22]. Please note that this transformation is actually a model-to-text transformation, since the employed feature model representation is technically based on XML, thus, XML code is produced. In line 1–3 all solutions of the model are queried, while in line 9–15 we iterate over these solutions and create a feature for each solution containing all of its alternatives as possible subfeatures (line 11–13). Thus, alternatives belonging to the same Solution are disjunct by default. In order to automatically derive conflicts between migration alternatives belonging to different Solutions, in line 20–36 we create a rule for each migration that is in conflict with another migration alternative. More specifically, a conflict between Alternatives implies that those alternatives are not compatible to each other (line 27–29). The generated constraints correspond to the following expression, which states that selection of a implies that a_1, ..., a_n are not valid anymore, i.e., they are in conflict with a, more formally:

$$a \Rightarrow \neg a_1 \wedge \neg a_2 \wedge \ldots \wedge \neg a_n$$

Once the feature model is generated, it can be loaded, displayed and edited by the FeatureIDE plugin [27] for Eclipse.

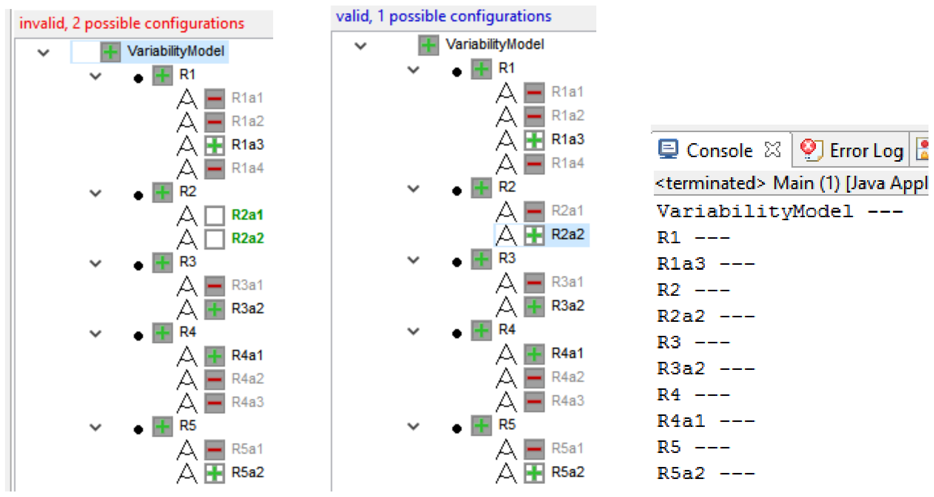
Listing 1.1: Fragment of the *WMM2FM* transformation

```
[%
var solutionModel := SolutionM.allInstances().at(0);
var allSolutions := solutionModel.solutions; %]
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel chosenLayoutAlgorithm="1">
  <struct>
    <and abstract="false" mandatory="true" name="SolutionModel">
[%
for (s in allSolutions) { %]
      <alt mandatory="true" name="[%=s.name%]">
        [% for (a in s.alternatives) { %]
          <feature mandatory="true" name="[%=a.name%]" />
        [% } %]
      </alt>
[% } %]
    </and>
  </struct>
  <constraints>
[%
for (s in allSolutions) {
  for (a in s.alternatives) {
    if (a.allConflicts.size() > 0) { %]
    <rule>
      <imp>
        <var>[%=a.name%]</var>
        <conj>
        [% for (conflict in a.allConflicts) { %]
          <not><var>[%=conflict.name%]</var></not>
        [% } %]
        </conj>
      </imp>
    </rule>
    [%
    }
  }
} %]
  </constraints>
  ...
</featureModel>
```

The feature model represents all possible configurations (i.e., migration solutions) for model migration. However, in order to create one specific m' a single configuration in the feature model must be selected. Since configurations can also be executed with FeatureIDE, it is possible to automatically migrate m to m' by attaching the needed migration actions for model migration directly to the alternatives in the feature model.

3.5 Configuration and Execution of Model Migration

As aforementioned, the modeler has to ultimately decide on a combination of valid options in the feature model, i.e., a configuration, to define a migration. In this process, the user is supported by the provision of constraints that restrict the number of valid solutions. In Fig. 7a a concrete but not yet finalized configuration taken by the user is shown. It is highlighted in the picture that the decision for R2 is not made yet, i.e, the modeler can still choose among the two available configurations (empty boxes shown in green). Once the user decides for a valid configuration (cf. Fig. 7b), this configuration can be executed. This means that all migration actions attached to the alternatives can be applied. Please note



```
VariabilityModel ---
R1 ---
R1a3 ---
R2 ---
R2a2 ---
R3 ---
R3a2 ---
R4 ---
R4a1 ---
R5 ---
R5a2 ---
```

(a) Building of configuration

(b) Concrete example of configuration

(c) Execution log of the configuration

Fig. 7: Configuration and Execution of Model Migration

that discussing the actual migration process is outside the scope of this paper, however migration actions elaborated in our previous work [17] are adequate to be reused in this approach. As shown in Fig. 7c an exemplary execution log is provided to show the potential of this approach when executed.

4 Discussion

Although the approach has been validated by considering representative examples only, early feedback provides interesting elements for outlining benefits and potential drawbacks. The idea of using feature models for representing the various alternatives simplifies the representation of explicit and relevant knowledge to be conveyed to the modeler. A manifest representation of alternatives, their conflicts, and how traceability cross-links them to the metamodel refactoring is a useful instrument for assisting the modeler in meeting her migration design decisions. On the contrary, model migration has always been based on individual and spontaneous processes prone to errors and inconsistencies. Therefore, shifting the responsibility of deciding which migration best fits the requirements from the migration program implementer to the modeler is greatly beneficial if properly supported.

To the best of our understanding, the major drawback of the approach is represented by the manual creation of the weaving model and the conflict representation. Although the automated generation of such artifacts is outside the scope of this paper, both the weaving model and the conflict representation we are confident that it can be obtained in an automated manner. In particular, we plan to revise our work from [25] (in which a corrupted conformance relationship is re-established by applying so-called repair actions that generate multiple solutions) to generate the weaving model instead of the concrete solutions. For identification of conflicts, starting from our work on dependent metamodel changes [3], we intend to generate admissible scheduling of migration actions. Both extensions are desirable and seem viable as no major technical obstacle is evident at this stage.

5 Related Work

There has been only little research on the unified management of multiple migration strategies in the modeling community so far. Thus, in the following, first more close approaches are compared to our approach, while in the latter approaches from other engineering domains are discussed.

The closest work to our approach has been proposed recently in [11], in which the authors propose a Variable Metamodel (VMM) during metamodel evolution. This metamodel unifies the concepts of different metamodel evolutions in a way that all models that need to be migrated conform to the VMM. In fact, a model is not migrated but matches the VMM in each evolution of the metamodel, which is in contrast to our approach since we provide possibilities to explore different migration alternatives how the model can conform to the latest version of the metamodel.

In [25] the authors introduce an approach to re-establish the conformance relationship between models and metamodels by manipulating the non-conforming model according to specific rules. Thus, multiple valid m' are being generated that are sorted and presented to the user according to quality criteria. However, each solution is presented as its own model in contrast to the approach proposed in this paper, which attempts to present variability in model migration in one single model to better cope with overlapping solutions.

In [15] variability is tackled by the provision of multiple alternative repair actions in order to repair a violated, i.e., non-conforming, model. However, the user has to decide on the lower level of repair actions, while in our approach the user decides on the level of models.

An approach which allows to define custom migration actions while still satisfying quality criteria, and thus tackles variability by user involvement, has been proposed in [19]. However, exploration of the different possibilities for migration is not part of their work.

In [9] the consistency restoration between different UML models, e.g., class and sequence diagrams, is addressed. To ensure consistency, an approach to automatically generated choices to repair inconsistent UML models is proposed. The tool lets the user explore alternative ways to fix inconsistencies in a UML model. However, the tool is limited to generate resolutions that only involve a change at a single location at a time.

The necessity of dealing with multiple alternatives arised also in other fields, including model merging and versioning and requirement engineering. In [5] the authors propose an approach to automatically merge different versions of a model according to user-definable consistency constraints. In fact, in case of inconsistencies the approach is able to inform the modeler about which model elements have to be changed. However, the approach focuses on merging different model versions into one model, while our approach is able to highlight different migration alternatives in one variability model.

Wieland et al. [30] present an approach for optimistic model versioning, meaning that conflicts do not have to be resolved immediately but rather when a decision can be made how to resolve them. The approach is able to accept conflicts and resolve them later in the process, by having the conflicted model elements annotated to reflect the modifications. However, the approach focuses on the simultaneous editing of models and arising conflicts, in contrast to our approach which deals with different strategies on how to migrate a model.

In [26] the authors propose an approach merge similar algebraic graph transformation (AGT) rules and generate a single rule with variability. Doing so, rule variants can be expressed in a compact manner.

In [10] partial models are introduced in order to let the designer to specify uncertain information by means of a base model enriched with annotations and first-order logic, which highlights the need for variability also in other engineering domains. In [23] the authors stress the need for uncertainty since the requirements engineering field it is common to have uncertainty in both the content and structure of the models. However, they do not cope with uncertainty by providing a number of different possible choices to resolve uncertainty.

In goal-oriented methodologies such as KAOS or i* [28] intentionality and variability aspects are also treated, but they are more widely used in early phases of a project, while the models considered in this paper are the central artifacts in the development process.

6 Conclusion and Future Work

In this paper, an approach for representing and visualizing different solutions for model migration in presence of metamodel changes has been proposed. Besides the capability to represent all possible alternatives in an intensional fashion, the approach permits the explicit representation of conflicts that easily arise in the migration process. In particular, the different options are represented in a weaving model between the model to migrate and the evolved version of the metamodel. In addition, it has been shown how to transform the weaving model into a feature diagram, a commonplace notation that can be used out-of-the-box. Among the advantages of the approach there is the traceability between the metamodel refactorings and the migrations alternatives, which provides a useful way to better grasp the rationale behind the migration actions.

The presented approach suggests further developments. In particular, the idea of automating the generation of the weaving models seems viable according to our previous work [25]. The generation of conflicts is also another aspect we intend to investigate starting from our previous work on the management of dependent changes [3]: an opportunity to harness is given by analyzing how dependencies among metamodel changes can give place to scheduling of model migration actions.

References

1. Batory, D.: Feature Models, Grammars, and Propositional Formulas. In: Obbink, H., Pohl, K. (eds.) Software Product Lines, LNCS, vol. 3714, pp. 7–20. Springer (2005)
2. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the Large and Modeling in the Small. In: Amann, U., Aksit, M., Rensink, A. (eds.) Model Driven Architecture, LNCS, vol. 3599, pp. 33–46. Springer (2005)
3. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: Managing dependent changes in coupled evolution. In: Theory and Practice of Model Transformations, pp. 35–51. Springer (2009)
4. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: Proc. of EDOC. pp. 222–231. IEEE (2008)
5. Dam, H.K., Egyed, A., Winikoff, M., Reder, A., Lopez-Herrejon, R.E.: Consistent merging of model versions. Journal of Systems and Software (2015)
6. Di Ruscio, D., Iovino, L., Pierantonio, A.: Coupled evolution in Model-Driven Engineering. IEEE Software 29(6), 78–84 (2012)
7. Di Ruscio, D., Iovino, L., Pierantonio, A.: Evolutionary Togetherness: How to Manage Coupled Evolution in Metamodeling Ecosystems. In: ICGT. vol. 7562, pp. 20–37. Springer (2012)
8. Di Ruscio, D., Paige, R.F., Pierantonio, A.: Guest editorial to the special issue on success stories in model driven engineering. Science of Computer Programming 89, 69–70 (2014)
9. Egyed, A., Letier, E., Finkelstein, A.: Generating and Evaluating Choices for Fixing Inconsistencies in UML Design Models. In: 23rd IEEE/ACM International Conference on Automated Software Engineering. pp. 99–108 (Sept 2008)
10. Famelis, M., Salay, R., Chechik, M.: Partial models: Towards modeling and reasoning with uncertainty. In: Proc. of ICSE. pp. 573–583 (June 2012)
11. Font, J., Arcega, L., Haugen, O., Cetina, C.: Addressing Metamodel Revisions in Model-based Software Product Lines. In: Proc. of the 2015 ACM SIGPLAN Int. Conf. GPCE. pp. 161–170. ACM (2015)
12. Garcés, K., Jouault, F., Cointe, P., Bézivin, J.: Managing model adaptation by precise detection of metamodel changes. In: Model Driven Architecture-Foundations and Applications. pp. 34–49. Springer (2009)
13. Herrmannsdoerfer, M.: COPE A Workbench for the Coupled Evolution of Metamodels and Models. In: Malloy, B., Staab, S., van den Brand, M. (eds.) SLE 2010. LNCS, vol. 6563, p. 286295. Springer (2011)
14. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of mde in industry. In: Proc. of the ICSE. pp. 471–480. ACM (2011)
15. Körtgen, A.T.: New Strategies to Resolve Inconsistencies between Models of Decoupled Tools. In: 3rd Workshop on Living with Inconsistencies in Software Development, Bd. vol. 661, pp. 21–31 (2010)

16. Kurtev, I., Bzivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA'2002 Federated Conferences, Industrial track (2002)
17. Kusel, A., Etzlstorfer, J., Kapsammer, E., Retschitzegger, W., Schwinger, W., Schönböck, J.: Consistent Co-Evolution of Models and Transformations. In: Proc. of the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, Ottawa, Canada (10 2015)
18. Lientz, B.P., Swanson, E.B.: Software maintenance management. Addison-Wesley (1980)
19. Mantz, F., Taentzer, G., Lamo, Y.: Well-formed Model Co-evolution with Customizable Model Migration. Electronic Communications of the EASST 58 (2013)
20. Parsons, J., Wand, Y.: Using objects for systems analysis. Commun. ACM 40(12), 104–110 (1997)
21. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.: Model Migration with Epsilon Flock. In: ICMT, LNCS, vol. 6142, pp. 184–198. Springer (2010)
22. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.A.: The Epsilon Generation Language. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture Foundations and Applications, LNCS, vol. 5095, pp. 1–16. Springer (2008)
23. Salay, R., Chechik, M., Horkoff, J., Di Sandro, A.: Managing requirements uncertainty with partial models. Requirements Engineering 18(2), 107–128 (2013)
24. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (Feb 2006)
25. Schönböck, J., Kusel, A., Etzlstorfer, J., Kapsammer, E., Schwinger, W., Wimmer, M., Wischenbart, M.: CARE – A Constraint-Based Approach for Re-Establishing Conformance-Relationships. In: Proc. of the APCCM (2014)
26. Strüber, D., Rubin, J., Arendt, T., Chechik, M., Taentzer, G., Plger, J.: RuleMerger: Automatic Construction of Variability-Based Model Transformation Rules. In: Fundamental Approaches to Software Engineering. LNCS, vol. 9633. Springer (2016)
27. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming 79, 70 – 85 (2014)
28. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. pp. 249–262. IEEE (2001)
29. Wagelaar, D., Iovino, L., Di Ruscio, D., Pierantonio, A.: Translational semantics of a co-evolution specific language with the EMF transformation virtual machine. In: ICMT, LNCS, vol. 7303, pp. 192–207. Springer (2012)
30. Wieland, K., Langer, P., Seidl, M., Wimmer, M., Kappel, G.: Turning Conflicts into Collaboration. Computer Supported Cooperative Work 22(2-3), 181–240 (2013)