Relaxing Metamodels for Model Family Support

Sanaa Alwidian
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
salwidia@uottawa.ca

Daniel Amyot
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
damyot@uottawa.ca

Abstract—A model family regroups related models that vary along some dimension such as time or product in (software) product lines. A model family can be captured as a "150% model" that merges the family members, while enabling the extraction of the individual models. However, this 150% model may no longer conform to the original metamodel of the family members. This paper focuses on the evolution of a language's metamodel to accommodate both the original models and the 150% model. In particular, it aims to define a technique that minimally relaxes the metamodel constraints related to multiplicities of attributes and association ends in order to enable conformance. The paper uses illustrative examples from two modeling languages (UML class diagrams and the Goal-oriented Requirement Language) to describe the problem and to explore potential approaches for metamodel relaxation. While early results are promising, there are important challenges remaining to balance conflicting forces at play, e.g., having a minimal relaxation (such that existing analysis techniques can be easily adapted for the 150% model) and predicting where relaxation is needed in the metamodel

Keywords— Conformance; constraint relaxation; evolution; metamodel; model-driven engineering; model family.

I. Introduction

Over the last years, Model-Driven Engineering (MDE) has gained significant importance and popularity as a matured discipline that has been applied in a wide array of academic and industrial domains [1]. At the core of MDE is the use of two interrelated artefacts: metamodels and models. A model is said to conform to (or instantiates) its metamodel if the former obeys the rules and constraints imposed by the metamodel. In a sense, a metamodel describes the abstract syntax and the static semantic of modeling languages [2]. In addition, Metamodels impose a significant number of rules (e.g., the concepts to be used) and constraints (e.g., multiplicities) that govern how models should be created.

MDE (meta)models must be managed as they usually *evolve* over time. In the literature, a well-addressed aspect of MDE evolution is the *metamodel evolution and model co-evolution* [3]-[7]. In these approaches (Fig. 1), model evolution from M to M' is carried out *after* metamodel evolution from MM to MM'. However, to the best of our knowledge, there is a lack of approaches that attempt to evolve/change metamodels in response to model evolution. We initially refer to this context as the *model-triggered metamodel evolution* problem. The general description of this problem (Fig. 2) is characterized as: *If a model M (that conforms to a metamodel MM) evolves over time, resulting in a new model version M' that is no longer conforming to the original metamodel MM, how should we extend MM (ideally with the least amount of changes) into MM', in order for M' to be conform to MM'?*

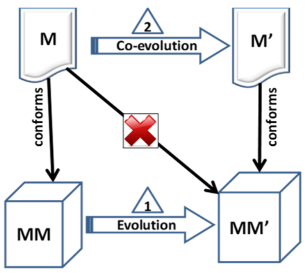


Fig. 1. Metamodel (MM) evolution and model (M) co-evolution problem

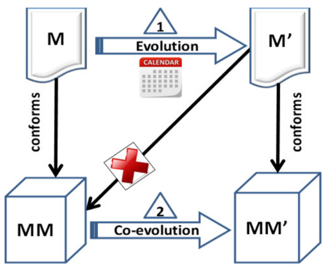


Fig. 2. Model-triggered metamodel (MM) evolution problem (general)

A *model family* regroups a set of related models that evolve, for example, over *time*. Another common source of model families is found in *(software) product lines*, where different *versions/variations* of a product can exist without necessarily being caused by some evolution over time. A model family can be captured as a *150% model* which consists of the union of the model elements from all valid family members [8], aggregated in a way that enables the extraction of an individual model. Such 150% models are mainly used in the context of model-based product lines, where products are derived using negative variability by removing artefacts from the family model according to a given feature configuration [9].

The general problem in Fig. 2 allows for individual models to be non-compliant to the original metamodel, MM (e.g., by using a class or association not supported in the metamodel). However, for model families, this paper assumes that *all family members conform to the same metamodel*. In *this* context, one important observation is that even if each family member conforms to the original metamodel, there is still *no* guarantee that the 150% model aggregating all individual models will conform to the original metamodel. In this case, we end up having an invalid/ill-formed 150% model that requires us to revisit the original metamodel and evolve it in a way that re-establishes conformance, such that family members *and* the 150% models comply to the evolved metamodel. Another interesting observation here is that having new members in a model family does not require additions, deletions, or modifications of concepts to the original metamodel. This means that the resulting metamodel needs to be only *relaxed* (in terms of its constraints) rather than extended. In a model family context, the general problem (Fig. 2) can be substantially simplified and specified as: *If models $M_0 ... M_n$ (that conform to a* *metamodel MM) are aggregated, resulting in a new model M_{150} that is no longer conforming to the original metamodel MM, how should we extend MM (ideally with the least amount of changes) into MM_{150}, in order for M_{150} as well as $M_0..M_n$ to be conform to MM_{150}*? Fig. 3 illustrates this problem.

This paper illustrates and discusses this model family-specific metamodel evolution problem (Fig. 3) and proposes an initial solution called **Me**tamodel **Re**laxation (MeRe), while highlighting remaining challenges. The remainder of this paper is organized as follows: Section II discusses the motivation and scope of this work. Section III provides three illustrative evolution scenarios. The proposed approach is discussed in Section IV. Section V discusses how analysis can be conducted on 150% models. Related work is presented in Section VI. Finally, Section VII concludes and provides our future plans towards solving the model family-specific problem.

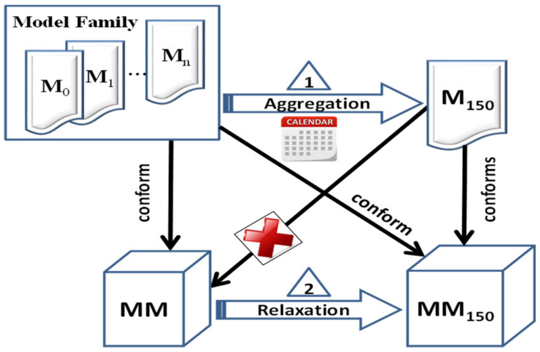


Fig. 3. Model family-specific metamodel evolution problem

II. Motivation and Scope

Our main motivation comes from our experience using goal models for modeling regulations, in collaboration with Transport Canada. Regulators have regulations for different types of parties (say, airports of different sizes). Trying to model these different regulations, for analysis or compliance purposes, we ended up requiring different goal models [9]-[11]. We tried to capture all variants with one model, using the Goal-oriented Requirement Language (GRL) [12], to minimize maintenance problems, but ended up with metamodel-conformance issues because the language did not allow us to capture the family with one unified model. This is an issue along the product dimension, which is not limited to GRL but is common to most modeling languages. We then realized that similar problems occur along the time dimension when a model evolves. If a product has many versions over time, and if we want to analyze all versions (say, before releasing a patch that would affect all versions), a model representing the family (i.e., a 150% model) becomes necessary to allow reasoning about all members at once instead of reasoning about each member individually. However, the concern here is not the construction of the 150% model itself. Rather, the challenge emerges when a 150% model violates conformance with the original metamodel, MM. In this case, we need to relax MM into MM_{150} to ensure that this 150% model is representable. A minimal relaxation is desirable in this context so as to minimize modifications to existing tools and analysis approaches.

The main focus of this work is on the challenge of inferring the metamodel of a model family from the structure of the metamodel of its members through a minimal set of constraint relaxations. We are not dealing with the more general problem of inferring a metamodel from a collection of models that conform to different metamodels (i.e., our member models use the same language). Also, we are not dealing with a dynamic metamodel co-evolution upon changes; generating a new metamodel each time there is a new member added to the family is not practical, as this would imply developing new tools (for producing, editing, analyzing, and transforming 150% models) each time. We are also not looking at language-specific solutions (e.g., through the use of metadata and user links in GRL). The long-term challenge is hence to *predict* the locations where metamodel relaxations are needed, without relaxing too much so existing tools and techniques would require a minimum of adaptation. Our hope is to develop tools for the relaxed language only *once*. This research aims to answer the following research questions:

- **RQ1)** How can we minimally relax a metamodel to support the aggregation of members of a model family in a way that enables the generation of all (and only) individual members?
- **RQ2)** How can we predict where relaxation is needed (relaxation points) in the original metamodel, for all potential 150% models?
- **RQ3)** What kinds of analyses can be conducted on model families to reason on all members of a 150% model?

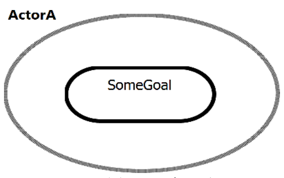
The following section provides concrete evolution scenarios further illustrating the problem.

III. Sample Evolution Scenarios

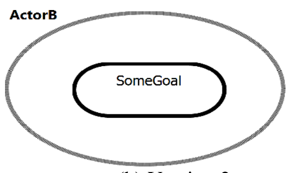
This section provides three illustrative examples from two modeling languages: the Goal-oriented Requirement Language (GRL) and UML class diagrams. These examples serve as evidence to justify the need for relaxing some of the metamodel multiplicities and other constraints (RQ1).

A. *Example One: Different GRL Actors with the Same Goal*

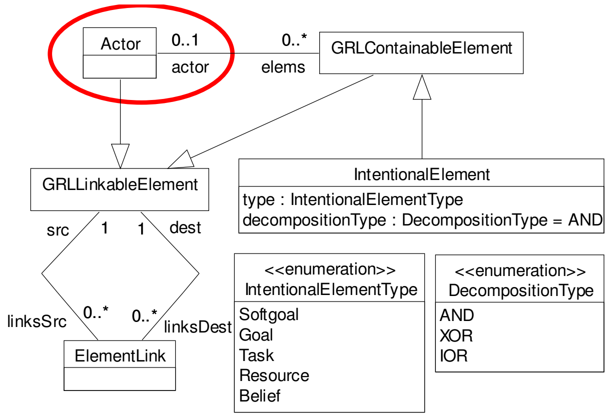
In this section, an evolution scenario for simple GRL models is given. GRL is part of the User Requirements Notation standard [12], [13]. As shown in Fig. 4.a, the first version of the GRL model represents an actor (ActorA) containing an intentional element of type goal (SomeGoal). Let us assume that this model evolves over time, such that SomeGoal is moved from ActorA to ActorB (or when ActorA is replaced by ActorB). The evolved version is shown in Fig. 4.b. Each of the model versions, separately, conforms to the metamodel excerpt represented in Fig. 4.c, as an intentional element may be included by 0 or 1 actor. However, if we want to aggregate both models into one 150% model to represent their model family, the resulting 150% model will not conform to the current metamodel. This is because an intentional element (SomeGoal) cannot be contained by two actors. This violates the multiplicity constraint circled in the metamodel excerpt (Fig. 4.c).



(a) Version 1



(b) Version 2



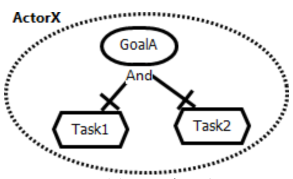
(c) Extract of the GRL metamodel

Fig. 4. First GRL example: SomeGoal moves from ActorA to ActorB, with relevant GRL metamodel elements

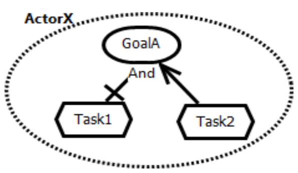
One way to re-establish conformance is to relax the multiplicity constraint of the actor association end to be (0...*). The purpose of this relaxation is to allow different actors in different model versions to contain the same intentional elements. Such new and relaxed MM_{150} metamodel would hence allow the 150% model (M_{150}) and the individual family members (versions 1 and 2) to be conform.

B. *Example 2: GRL Goals with Multiple Links*

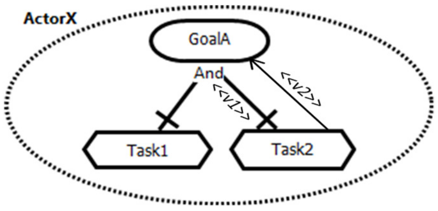
The example in Fig. 5 shows another evolution scenario in GRL. In the first version (Fig. 5.a), a goal (GoalA) is decomposed into two intentional elements of type task (Task1 and Task2). Let us assume this model evolves such that the type of link that associates GoalA with Task2 changes from a decomposition link to a contribution link (as shown in Fig. 5.b). Decompositions and contributions are two different sub-types of ElementLink in the metamodel of Fig. 4.c.



(a) Version 1



(b) Version 2



(c) 150% model

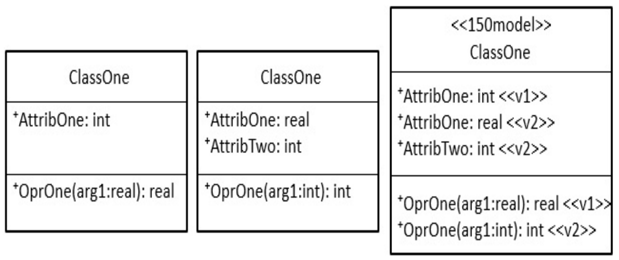
Fig. 5: (a) Version 1: GoalA and Task2 with decomposition link (b) Version 2: GoalA and Task2 with Contribution Link (c) 150% model that regroups both models

The 150% model The 150% model that aggregates both models is shown in Fig. 5.c. However, this 150% model cannot be represented by the original metamodel because of an existing (OCL) constraint stating that any pair of IntentionalElements can be connected by at most one ElementLink (Fig. 4.c.). Hence, it is not allowed to have both a decomposition link and a contribution link between GoalA and Task2. Therefore, to support the representation of such 150% models, the OCL constraint of the original metamodel MM needs to be relaxed (or removed) in the evolved metamodel MM150 to allow more than one element link to connect the same pair of intentional elements in the aggregated M_{150} family model (Fig. 3). In addition, as shown in Fig. 5.c, we need to distinguish the elements in the 150% model that belong to different versions of models. We annotate these elements with information about version numbers (for example <<v_1>>) to indicate that this particular element is part of *only* version 1 of the model. The version-based annotations are of particular importance to facilitate the extraction of individual models from the 150%. The absence of such annotation means that all versions (i.e., all members of the family) include that model element. Annotations are directly supported with "metadata" attached to any model element in GRL. However, should an annotation mechanism not be available in the source modeling language, the relaxed MM150 metamodel may also need to be extended to include such a mechanism.

C. *Example 3: UML Attributes with Multiple Types*

This third evolution scenario This third evolution scenario focuses on UML class diagrams [14] Let us assume that the first version (Fig. 6.a) of class ClassOne has an attribute AttribOne of type int, as well as an operation OprOne, with argument arg1 of type real, that returns a parameter of type real. This class evolves to version 2 (Fig. 6.b) such that the type of AttribOne becomes real instead of int, and the operation' parameter and return data types become int (instead of real). These two versions are aggregated into a 150% model, illustrated in Fig. 6.c, where version annotations are also used on attributes and operations.

Having multiple operations with the same names but different argument types is allowed in UML. However, UML enforces stronger constraints on attribute names. The resulting 150% model violates the UML standard metamodel since the latter does not support the representation of one attribute with multiple data types. In order to re-establish conformance, the multiplicity constraint related to attributes could be relaxed in UML such that attributes would be allowed to have many types instead of only one.



(a) Version 1 (b) Version 2 (c) 150% model

Fig. 6. Two ULM classes and their resulting 150% model

D. Observations

Based on the three previous evolution scenarios, there are some potential partial answers that can be observed. Regarding the first part of RQ1, i.e., "How can we minimally relax a metamodel to support the aggregation of members of a model family?", we conjuncture that this can be done through relaxing multiplicity constraints of association ends and attributes, as well as external (OCL) constraints on the metamodel. Constraint relaxation also ensures that the original individual models (family members) remain conform to the new metamodel. Regarding the second part of the question, i.e., "…in a way that enables the generation of all (and only) individual members?", we conjuncture that the explicit tagging of versions in the aggregate model enables the reconstruction of the original models (the family members) and prevents the generation of other hybrid models

IV. Metamodel Relaxation for Model Families

This section discusses our proposed approach, MeRe, for metamodel relaxation triggered by model evolution in a model family context. MeRe proposes four phases for enabling metamodel relaxation: model aggregation, change detection, non-conformance detection, and metamodel relaxation inference. An overview of the MeRe approach is given in Fig. 7 and its details are discussed in the next sections.

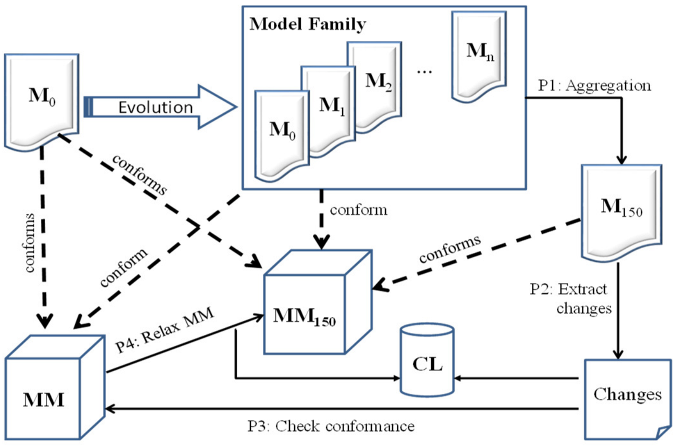


Fig. 7. MeRe approach

A. Phase 1: Model Aggregation

The goal of this phase is to aggregate the various models of a model family into one single model, M_{150}. Let M_0 be the initial version of a model that conforms to *MM* and evolves over time or across product variants into several versions, M_1, M_2, M_3, etc. The set of versions, *V*, is $V= \{M_0, M_1, M_2 ... M_n\}$, (where *n* is the number of modified versions of a particular model). M_{150} is the *union of all elements* in all models of *V*. The union of models at an element level is defined as follows. M_x is a tuple $\{E^x , L^x\}$ with a set of elements $E^x=\{E^x{}_1, E^x{}_2, E^x{}_3,...,E^x{}_i\}$, such that $E^x{}_1$ identifies the first element of version *x* of the model, and so on. M_x also contains a link set L^x, where $L^x \subseteq E^x \times E^x$. $L^x{}_{\text{a-b}}$ denotes a link between elements *a* and *b* of model M_x. The union of two models M_x and M_y is defined in equation 1:

$$Mx \cup My = \{\{Ex \cup Ey\}, \{Lx \cup Ly\}\}, 0 \leq x \leq n, 0 \leq y \leq n \qquad (1)$$

Therefore, M_{150} is the union of all elements and all links of models in *V*, as defined in equation 2:

$$M_{150} = \bigcup_{i=0}^{n} Mi \qquad (2)$$

B. Phase2: Change Detection

In this phase, changes among models are detected through the use of M_{150}, whose elements can now be extended with a Δ annotation. To illustrate the basic idea of this phase, we take as an example, the evolution scenario captured in Fig. 5, where we consider each model version as a *set* of elements. For instance, version 1 of the model consists of: GoalA, Task1, Task2, a decomposition link between GoalA and Task1 and another decomposition link between GoalA and Task2. We refer to these elements and links as: $E^1{}_1$, $E^1{}_2$, $E^1{}_3$, $L_{E^1{}_1\text{-}E^1{}_2}$ and $L_{E^1{}_1\text{-}E^1{}_3}$, respectively. Therefore, version 1 of the model can be denoted as: $M_1= \{E^1{}_1, E^1{}_2, E^1{}_3, L_{E^1{}_1\text{-}E^1{}_2}, L_{E^1{}_1\text{-}E^1{}_3}\}$. Following the same principle, version 2 of the model would be represented as $M_2= \{E^2{}_1, E^2{}_2, E^2{}_3, L_{E^2{}_1\text{-}E^2{}_2}, L_{E^2{}_1\text{-}E^2{}_3}{}^{\Delta}\}$, where the delta (Δ) denotes a change in the link type from M_1 to M_2 (for example, a decomposition link becomes a contribution link). This delta could be inferred by calculating the difference between M_1 and M_2, $Diff(M_1, M_2)$, using, for example, the approach proposed by Rivera et al. [14]. Since M_{150} is the union of models at the element level, then $M_{150}= \{E_1, E_2, E_3, L_{E1\text{-}E2}, L_{E1\text{-}E3}, L_{E^2{}_1\text{-}E^2{}_3}{}^{\Delta}\}$. Note that in M_{150}, we omit the superscript numbers (except for the last link) that refers to model versions to which elements and links belong. This is because these elements belong to both models M_1 and M_2. Therefore, there is no need to annotate them with the version number. The purpose of this phase is to detect and extract pairs of elements that have changed, denoted as E_i and $E_i{}^{\Delta}$. These pairs of changes are taken as an input to the next phase.

C. Phase 3: Non-Conformance Detection

At this phase, conformance between the original metamodel *MM* and M_{150} model is verified. This is done by checking if the co-existence of change pairs (obtained in phase 3) in the same model could cause a violation of association/attribute multiplicities or other external (OCL) constraints of *MM*. For instance, checking that two different links between exist the same pair of GRL intentional elements (e.g., $L_{E1\text{-}E3}$ and $L_{E1\text{-}E3}{}^{\Delta}$) would cause a conformance violation. If non-conformance is detected, phase 4 takes place.

D. Phase 4: Infer Relaxation to Metamodel

Based on the metamodel conformance violations detected in phase 3, the modeler is now able to decide on two things: the *extension type* and the *extension point*. Regarding the extension type, we have already discussed that in order to support model families (with large number of models), we only need to *relax* the metamodel internal constraints that are related to multiplicities of attributes and association ends and/or external well-formedness constraints. Regarding the extension point (precisely, the relaxation points), the current solution provided by this heuristic is local. That is, it provides insights about extension points related directly to the type of change detected in phase 2, for specific models. However, if a new model is added to M_{150}, phases 2 through 4 need to be repeated again to detect new changes and infer relaxation points in the metamodel. At this level, it is still challenging to predict the exact locations (i.e., points) of metamodel multiplicities that need to be relaxed, independently of the models in a family. We understand that we do not have to follow the naïve brute-force approach and relax all multiplicity constraints and external constraints in the metamodel. Instead, we need to identify a technique to predict where relaxation is needed in the metamodel, based on its structure and perhaps patterns of usages of the language. By this, we could answer *RQ2*.

As an initial solution however, we suggest that the pairs of changes detected in phase 2 along with the relaxation actions inferred in phase 4 can be profiled in a separate file, called *Change Log (CL)*, such that the CL will contain tuples in the form of <*ChangeType, RelaxationType*>. Now, when a new model emerges, we can compare changes introduced by this new model with *ChangeTypes* stored in CL. If any similarity is detected, we can infer the type and point of relaxation needed in the metamodel without going through phase 3 and 4. Alternatively, based on a large collection of models, relaxation points could be discovered empirically.

V. Analysis Over 150% Models

This section is dedicated to the exploration of *RQ3*. As mentioned previously, 150% models represent the union of all members of a model family. 150% models do not only merge the family members, but also enable the extraction of the individual, original models. To facilitate model extraction, we suggest that elements of the 150% model be annotated with information about the possible variations in a model family, such as version identifiers as suggested by Shamsaei et al. [11] or presence conditions and meta-expressions as presented by Czarnecki and Antkiewicz [8]. The later annotations are defined in terms of features and feature attributes from a feature model, and can be evaluated with respect to a feature configuration, for instance, to indicate whether or not a particular element has to be part of all or some models. That is, a 150% model can be instantiated based on feature configurations (these instances could be products, systems, regulations, etc.).

The extraction of a particular individual model from the 150% model could be important for analysis purposes. For example, instead of examining models, from years 2007 to 2017, all at once, a decision maker could be interested to view one model only (for example, the model of 2013), to conduct different kind of analysis over it. In addition, the 150% model can be viewed as a single generic model that combines old (i.e., legacy) models with the most recent models. This option would facilitate the comparison among legacy systems and current systems to infer reasoning about the current status of the system (for example, in the regulatory domain) and also to anticipate the future of such systems.

The reuse of analyses techniques and tools that already exist for the original models to the context of the 150% model is also challenging. For example, GRL models can be analysed through satisfaction propagation algorithms [16], but such algorithms would need to be adapted to cope with the relaxed metamodel of the 150% model. A naïve approach would be to extract all the models from the family and then use existing algorithms and tools on them. A more interesting approach would be to detect whether the metamodel relaxations warrant modifications to the analysis algorithms. Yet another research direction would be to consider “lifting” the algorithms, an approach proposed by Salay et al. for model transformations [17].

Inspired by the data analytics domain [18], we also envision conducting several types of analysis over a model family (through 150% models) to reason on all members of the family. Such analysis approaches include: 1) Descriptive analysis to describe the main aspects or features of the models being analyzed. For example, this may describe how a regulatory model in a particular year differs from the model of the next year. This allows one to make comparisons among different models. 2) Exploratory analysis to analyze models to find previously unknown relationships. This type of analysis is useful for discovering new connections and to provide future recommendations. 3) Inference analysis to infer information about large population of models based on a small sample of models. For example, examining compliance level using a small sample of models to explain how well the entire system complies with regulations and rules. 4) Predictive analysis to analyze current and historical (or legacy) models to predict future happenings of events. The essence of such approaches is to use data on some models to predict values for other models. 5) Causal analysis to figure out what will happen to one or more models when some model gets changed, e.g., to study the impact of changing one or more rule in a particular regulatory model on the behavior of other models in the family.

VI. Related Work

The literature suggests approaches to manage metamodel evolution through studying the impact of metamodel evolution on models and/or on operations (e.g., migration and transformations) and then adapting models (or operations) to their evolving metamodel (as illustrated in Fig. 1). To handle metamodel and model co-evolution, several approaches create difference models to calculate the difference with the last evolved metamodel, such as in [19]. Then, this difference model is used to derive automatic transformations for co-evolution. Sprinkle [3],[20] proposed a visual graph transformation-based language in order to specify model migration between two metamodel versions. Gruschko et al. [21], [22] classify primitive metamodel changes into non-breaking, resolvable, and irresolvable changes. Then, they provide automatic migration rules for non-breaking and resolvable changes. Cichetti et al. [23] go a step further and try to detect composite changes, e.g., extracting a class based on the difference between metamodel versions. To adapt to metamodel changes and migrate models, a number of high-level transformations were proposed in [24]. These transformations are based on a generic model that supports versioning for both models and metamodels. In addition, several approaches have been proposed to (semi-)automate transformation migration. In [25], Davide et al. present an approach for the coupled evolution of metamodels and transformations. This approach tries to assess the cost of a change and use this assessment to infer transformation evolution (for instance, whether to make a go/no-go decision to evolve transformations or not). These approaches are conceptually different from our approach. While they adapt models (or operations) to their evolving metamodel, MeRe investigates evolving metamodels in response to the evolution of models in the model family context.

Metamodels can also be extended through the concept of *profiles*. A well-known example is the UML.2x profile mechanism [14]. With profiles, new constructs (stereotypes), properties (tagged values) and modeling rules (constraints) are added to further restrict the metamodel's constructs and enforce the well-formedness of models of the domain-specific language. Similarly, Ecore (the metamodeling language of EMF) allows users to attach annotations on any element of a metamodel to capture additional information [26]. Unlike MeRe which relaxes a metamodel, these approaches either add new concepts to the original metamodel, or modify the language's validity constraints by further constraining them instead of relaxing their restrictions.

Model versioning approaches have also been proposed for model evolution. The approach of Alanen and Porres [27] is one of the earliest works on UML model versioning. In this approach, differences between model versions are detected by calculating the created, deleted, and changed elements, and then by matching the unique identifiers of these elements. Odyssey-VCS 2 is a version control system for UML models [28]. It controls versioning by using state-based differencing to detect elements between different versions of a model. The Adaptable Model Versioning (AMOR) is another model version control system proposed by Altmanninger et al. [29], which provides a mechanism for conflict detection between models by supporting definitions of conflict resolution policies. In addition, AMOR contains a recommender component that provides suggestions to users on how to resolve the detected conflicts. Finally, Aprajita and Mussbacher [30] explicitly extended the metamodel of GRL to document explicit changes (additions/deletions) of model elements to specific versions of a metamodel. Although a model family can then be captured, this approach is specific to one language and currently incomplete in the kinds of changes to versions it can accommodate. Although these approaches handle model evolution and track it through versioning, none of these approaches exploits model evolution as a trigger to evolve/extend the original metamodels to enable conformance.

To manage *uncertainty* in MDE, Famelis et. al. [31], [32] proposed the use of *partial models* as formal development artifacts to enable the abstraction, reasoning, visualization and manipulation of possible alternative models. In this approach, a *set* of alternative models with uncertainty are merged and captured with one model called a partial model. While the idea of capturing models in one partial model is close to our idea of merging models of a family in one 150% model. The context and the purpose of our work is still different. In a sense, we do the merge for the sake of representing model families and relax infer metamodels accordingly, while in [31] merging models is done to describe the observable behaviour of a system.

Czarnecki and Antkiewicz [8] proposed a template-based approach for mapping feature models to representations of *variability* in UML models. The authors describe the concept of superimposed variants to realize a negative variability, which corresponds to the 150% model used in our approach. However, the purpose of using 150% models is different in both approaches.

Finally, a variety of model evolution approaches have adopted the key ideas from *database schema evolution* approaches, which deal with the migration of database records to adapt to the evolution of the database schema. Details about schema evolution approaches are beyond the scope of this paper, but the interested readers can refer to the work of Rahm and Roddick [33], [34] While many of the approaches from schema evolution have been adapted for model evolution, one key consideration has yet to be fully addressed: if a database record evolves, or a new record is added that is not already defined in the database schema, then there is a need to evolve the schema to insure compatibility with the new record. This issue is analogous to the model-triggered metamodel evolution problem investigated in the paper.

VII. Conclusion and Future Work

This paper defines a technique that minimally relaxes metamodel constraints to support both the original models as well as the 150% model of a model family. The evolution scenarios illustrated in this paper suggest that in order to support model families, the metamodel constraints that need to be relaxed are mainly related to multiplicities of association ends and attributes, and to external well-formedness constraints (RQ1). Since our MeRe approach is solely based on light-weight metamodel relaxation (instead of heavy-weight extensions that require adding new concepts to the metamodel), its results are promising in terms of enabling conformance with evolved models with, ideally, as few changes as possible. However, ensuring minimal relaxation by predicting where relaxation is needed in the metamodel independently of the family members (RQ2) as well as evolving analysis approaches for exploiting and coping with the 150% model (RQ3) are still challenging issues that need to be addressed in our forthcoming work. We hence invite the research community to investigate solutions to these important problems.

References

[1] J. Whittle, J. Hutchinson, and M. Rouncefield. "The state of practice in model-driven engineering." IEEE software vol. 31(3) pp. 79-85, 2014.

[2] R. F. Paige, D. S. Kolovos, and F. Polack, "A tutorial on metamodelling for grammar researchers," Science of Computer Programming, vol. 96, pp. 396–416, December 2014.

[3] J. Sprinkle and G. Karsai, "A domain-specific visual language for domain model evolution," Journal of Visual Languages and Computing vol. 15(3-4), pp. 291–307, 2004.

[4] G. Taentzer, F. Mantz, and Y. Lamo, "Co-transformation of graphs and type graphs with application to model co-evolution," in ICGT, LNCS, vol. 7562, pp. 326–340. Springer, 2012.

[5] L. Rose, D. Kolovos, R. F. Paige, and F. Polack, "Model migration with Epsilon flock," in ICMT 2010, LNCS, vol. 6142, pp. 184–198. Springer, Heidelberg, 2010.

[6] H. Konig, M. Lowe, and C. Schulz, "Model transformation and induced instance migration: a universal framework," in SBMF 2011. LNCS, vol. 7021, pp. 1–15. Springer, Heidelberg, 2011.

[7] A. Kusel, J. Etzlstorfer, et al., "A Systematic Taxonomy of Metamodel Evolution Impacts on OCL Expressions," in Models and Evolution 2014, CEUR-WS, vol. 1331, pp. 2–11, 2014.

[8] K. Czarnecki and M. Antkiewicz, "Mapping features to models: a template approach based on superimposed variants," in GPCE 2005, LNCS, vol. 3676, pp. 422–437. Springer, Berlin, Heidelberg, 2005.

[9] A. Polzer, D. Merschen, G. Botterweck, A. Pleuss, J. Thomas, B. Hedenetz, and S. Kowalewski, "Managing complexity and variability of

a model-based embedded software product line," Innovations in Systems and Software Engineering, 8(1), pp. 35–49, 2012.

[10] A. Palmieri, P. Collet, and D. Amyot, "Handling regulatory goal model families as software product lines," in Advanced Information Systems Engineering-27th International Conference, LNCS, vol. 9097, pp. 181–196. Springer, 2015.

[11] A. Shamsaei, D. Amyot, A. Pourshahid, E. Yu, G. Mussbacher, R. Tawhid, E. Braun, and N. Cartwright, "An approach to specify and analyze goal model families," in 7th System Analysis and Modelling (SAM) Workshop, LNCS, vol. 7744, pp. 347–52. Springer, 2012.

[12] ITU-T, Recommendation Z.151 (10/12) User Requirements Notation (URN) - Language definition. Online: https://www.itu.int/rec/T-REC-Z.151/en

[13] D. Amyot and G. Mussbacher, "User Requirements Notation: The first ten years, the next ten years," Journal of Software, 6(5), pp. 747–768, 2011.

[14] OMG, Unified Modeling Language (UML), Version 2.5, formal/2015-03-01, 2015.

[15] J.E. Rivera and A. Vallecillo, "Representing and operating with model differences," in International Conference on Objects, Components, Models and Patterns, LNBIP, vol. 11, pp. 141-160. Springer Berlin Heidelberg, 2008.

[16] D. Amyot, S. Ghanavati, et al., "Evaluating goal models within the Goal-oriented Requirement Language," International Journal of Intelligent Systems (IJIS), 25(8), pp. 841–877, August 2010.

[17] R. Salay, M. Famelis, J. Rubin, A. Di Sandro, and M. Chechik, "Lifting model transformations to product lines," in: 36th International Conference on Software Engineering (ICSE 2014), pp. 117–128. ACM, New York, 2014.

[18] CI&T, The Four Types of Analytics. http://www.ciandt.com/card/four-types-of-analytics-and-cognition [online; accessed: April 22, 2017].

[19] A. Cicchetti, D. Davide, R. Eramo, and A. Pierantonio, "Meta-model differences for supporting model co-evolution," in Proceedings of the 2nd Workshop on Model-Driven Software Evolution (MODSE), pp. 1–10, 2008.

[20] J. M. Sprinkle, Metamodel driven model migration. Doctoral dissertation, Vanderbilt University, Nashville, USA, 2003.

[21] S. Becker, B. Gruschko, T. Goldschmidt, and H. Koziolek, "A process model and classification scheme for semi-automatic meta-model evolution," in 1st Workshop MDD, SOA und IT-Management (MSI), GI, GiTO-Verlag, pp. 35–46, 2007.

[22] B. Gruschko, D. Kolovos, and R. F. Paige, "Towards synchronizing models with evolving metamodels," in International Workshop on Model-Driven Software Evolution, paper 3, 2007.

[23] B.A. Cicchetti, D. Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in 12th International Enterprise Distributed Object Computing Conference (EDOC), pp. 222–231. IEEE Computer Society, 2008.

[24] J. Hößler, M. Soden, and H. Eichler, "Coevolution of models, metamodels and transformations," in Models and Human Reasoning, Wissenschaft und Technik Verlag, pp. 129–154, Berlin, 2005.

[25] D. R. Davide, I. Ludovico, and A. Pierantonio, "A methodological approach for the coupled evolution of metamodels and ATL transformations," in ICMT 2013: Theory and Practice of Model Transformations, LNCS, vol. 7909, pp. 60–75. Springer, Berlin, Heidelberg, 2013.

[26] P. Langer, K. Wieland, M. Wimmer, and J. Cabot, "EMF profiles: a lightweight extension approach for EMF models," Journal of Object Technology, vol. 11, no.1, pp. 1–29, April 2012.

[27] M. Alanen and I. Porres, "Version control of software models," in Advances in UML and XML-Based Software Evolution, Chapter III, pp. 47–70. Idea Group Publishing, 2005.

[28] H. Oliveira, L. Murta, and C. Werner, "Odyssey-VCS: a flexible version control system for UML model elements," in Proc. 12th Int. Workshop on Software Configuration Management, pp. 1–16. ACM, 2005.

[29] K. Altmanninger, G. Kappel, et al., "AMOR – towards adaptable model versioning," in 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM'08), Workshop at MODELS'08, Toulouse, France, 2008.

[30] Aprajita and G. Mussbacher, "TimedGRL: Specifying goal models over time," in 6th Int. Model-Driven Requirements Engineering Workshop (MoDRE), pp. 125–134. IEEE CS, 2016. doi:10.1109/REW.2016.035.

[31] M. Famelis, S. Ben-David, M. Chechik, and Rick Salay. "Partial models: A position paper." In Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation, p. 1. ACM, 2011..

[32] M. Famelis, R. Salay, and M. Chechik. "Partial models: Towards modeling and reasoning with uncertainty". In proceedings of ICSE, pp. 573–583, 2012.

[33] E. Rahm and P.A. Bernstein, "A survey of approaches to automatic schema matching," VLDB Journal, vol. 10, no. 4, pp. 334–350, 2001.

[34] J. F. Roddick, "A survey of schema versioning issues for database systems," Information and Software Technology, vol. 37, no. 7, pp. 383–393,1995.