Multi-Model Evolution through Model Repair

Patrick Stünkel[1], **Harald König**[2, 1], **Adrian Rutle**[1], **and Yngve Lamo**[1]
[1]Western Norway University of Applied Sciences, Bergen, Norway
[2]University of Applied Sciences FHDW Hannover, Hanover, Germany

ABSTRACT Design and development of complex software systems usually comprises multiple inter-related models, i.e. abstract representations of certain aspects of the underlying system. The relations between these models induce global consistency conditions which the models collectively must fulfill. At the same time, these models are subject to frequent changes, and as a result, maintaining their global consistency over time becomes an important issue in model management in general and Model-Driven Software Engineering in particular.
In this paper, we present a comprehensive feature model providing an overview of the current state of the art of model management. In this feature model, we further identify the central role of model repair as an implementation pattern for (multi-)model evolution.

KEYWORDS Model Management, Model Co-Evolution, Multi-Modeling, Model Synchronization, Consistency Maintenance, Model Repair, Bidirectional Transformations (bx), Update Propagation, Model Migration, Feature Model.

1. Introduction

In Model-Driven Software Engineering (MDE), *models* are the primary artifacts in the software development process. This promises several benefits, e.g. higher efficiency due to working on a high level of abstraction, improved involvement of domain experts via Domain Specific Languages (DSLs), self-documenting artifacts and many more. Developing a *complex* system implies *multi*-modeling; i.e., the various aspects of the system are represented as inter-related artifacts, which are subject to ongoing changes. Hence, researchers have identified *evolution* as an important requirement for MDE (Paige et al. 2016; Lämmel 2014): A *local* model modification (e.g. due to refinement, refactoring, new or obsolete requirements, etc.) not only results in an update to that particular model but also induces *global* modifications on related models such that the final result is considered *consistent*, i.e. the collection of models *evolves* together.

Model evolution has been studied from different angles in different communities using different terminology: *coupled transformations* (Lämmel 2014), *multi viewpoint modeling* (Romero et al. 2009), *metamodel/model co-evolution* (Gruschko et al. 2007), *model migration* (Narayanan et al. 2009), *view maintenance* (Diskin et al. 2012), *megamodeling* (Favre & NGuyen 2005), *macromodeling* (Salay et al. 2009), *bidirectional transformations (bx)* (Czarnecki et al. 2009), *inter-model consistency management* (Diskin et al. 2011), *model synchronization* (Hermann et al. 2011), and *model repair* (Macedo et al. 2017).

The main contribution of this paper is a *feature model*, which consolidates the findings and classifications from recent surveys such as (Anjorin et al. 2019; Cicchetti et al. 2019; Hidaka et al. 2016; Hebig et al. 2017) in the above-mentioned domains. It is based on the significant work published in the model repair survey by Macedo et. al. (Macedo et al. 2017) comprising an extended and more refined presentation with a focus on multi-modeling. The feature model clearly separates between the different aspects of model management, namely representation of *models*, *changes*, *consistency*, *multi-models*, *formats*, and *repair*. This enables comparison of tools and approaches from originally separated research domains, e.g. multi-viewpoint modeling and bx. Furthermore, we motivate the importance of model repair as a *universal implementation pattern* for model management operations (Diskin 2009).

JOT reference format:
Patrick Stünkel, Harald König, Adrian Rutle, and Yngve Lamo. *Multi-Model Evolution through Model Repair.* Journal of Object Technology. Vol. 20, No. 1, 2021. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2021.20.1.a2

Outline This paper can logically be divided into two parts: The first sections build up the required background on model management and evolution starting with motivating examples (section 2), explaining the core concepts (section 3), and eventually focusing on model repair locally (section 4) and globally (section 5) as a core ingredient for evolution. Section 6 initiates the second half of the paper, where we throughout several subsections present our main contribution: the feature model. Section 6.9 exemplifies an application of the feature model to a selection of existing model management tools, section 7 gives a summary of valuable insights gathered throughout our literature study, and finally, section 8 concludes with an outlook on future work.

2. Multi-Model Management Scenarios

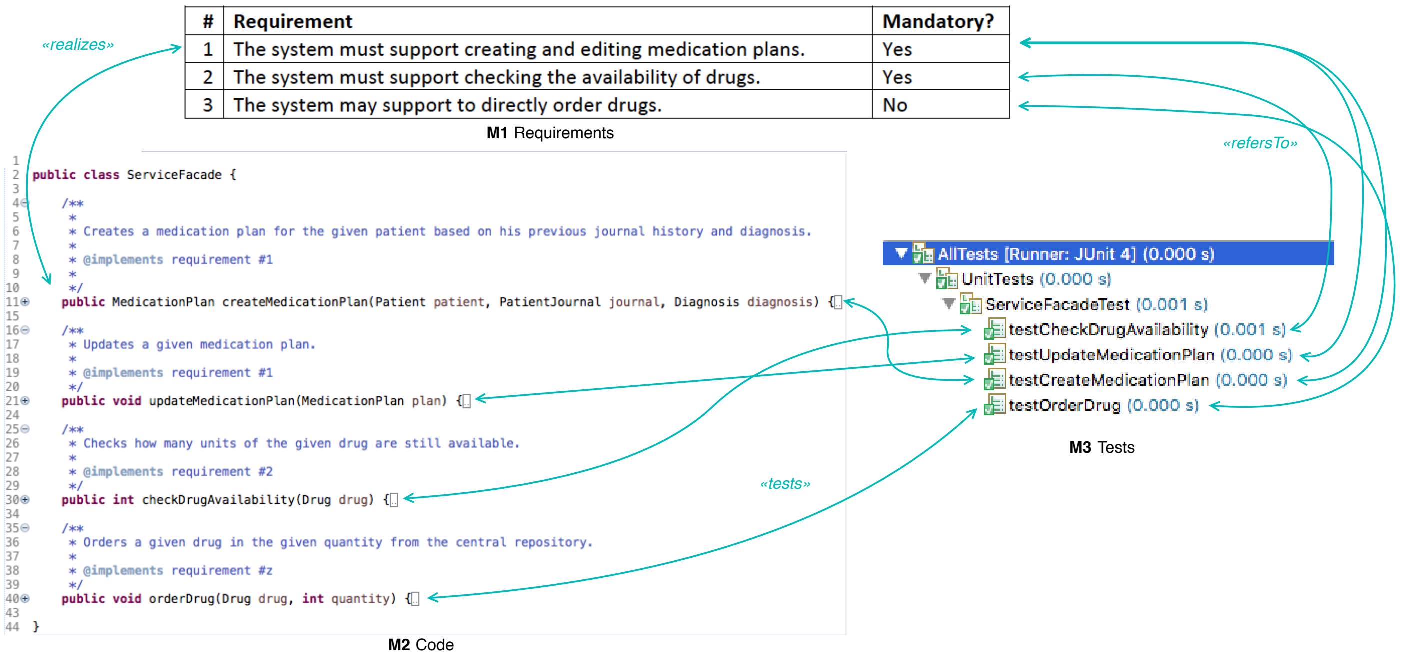
First, we want to introduce models, their relationships and important aspects such as correctness, ambiguity, incrementality etc. along a few motivating examples. Within MDE it is common to adopt the imperative of "everything-is-a-model" (Stevens 2020; Brambilla et al. 2017; Bézivin 2005). Thus, we treat all artifacts ranging over requirements, designs, drawings, source code, test cases etc. as *models*, i.e. abstract representations of (parts of) a real world software system. Our first toy scenario, depicted in Fig. 1 comprises three models: A list of *requirements* (M1), java *code* (M2), and unit *tests* (M3). Note the (cyan-colored) links depicting relations between elements across their model boundaries. These links have been referred to under different names in the literature: *Morphisms* (Bernstein 2003), *mappings* (Bernstein 2003), *traces* (Aizenbud-Reshef et al. 2006), *traceability links* (Feldmann et al. 2019), *commonalities* (Klare & Gleitze 2019) or *correspondences* (Stünkel et al. 2018).

Furthermore, let us consider a requirement:

CR-1 Every method, which realizes a requirement must have an associated test case and for every mandatory requirement there has to be at least one test case.

The requirement **CR-1** is an example of a *consistency rule* and they are a common phenomenon in model management, see e.g. (Feldmann et al. 2019). Such rules determine what constellations of models and inter-relations are considered *valid*. Thus, changing a model requires caution: Multiple models have to (co)-evolve simultaneously to guarantee consistency.

As an example for such an evolution scenario in Fig. 1, assume that a new mandatory requirement is added to M1. This obviously invalidates **CR-1** since there is no unit test for the new requirement. There are multiple possibilities to resolve this inconsistency: (1) adding a test stub, (2) letting an already existing test case refer to the new requirement, (3) making the new requirement optional, or (4) deleting the new requirement again. Option (4) is the least desirable one since it contradicts the idea of evolution by reverting the original change, whereas option (1) may be the most desirable solution since it is the "least intrusive" modification. However, in general, a tool cannot know the intention of the user. For instance, there may be situations where the user prefers the more intrusive option (2) or even option (3) —i.e. revising the original change. Furthermore, it is important to note that some of these options come in (possibly infinitely) many variations, e.g. there are almost unlimited choices of what the concrete *method body* of the test stub should be. Many repair approaches address this "disambiguation-challenge" by asking the user during the process or by working with default values. To make the situation even more interesting, assume that now someone adds a method realizing the new requirement. This also renders **CR-1** inconsistent because the method is missing

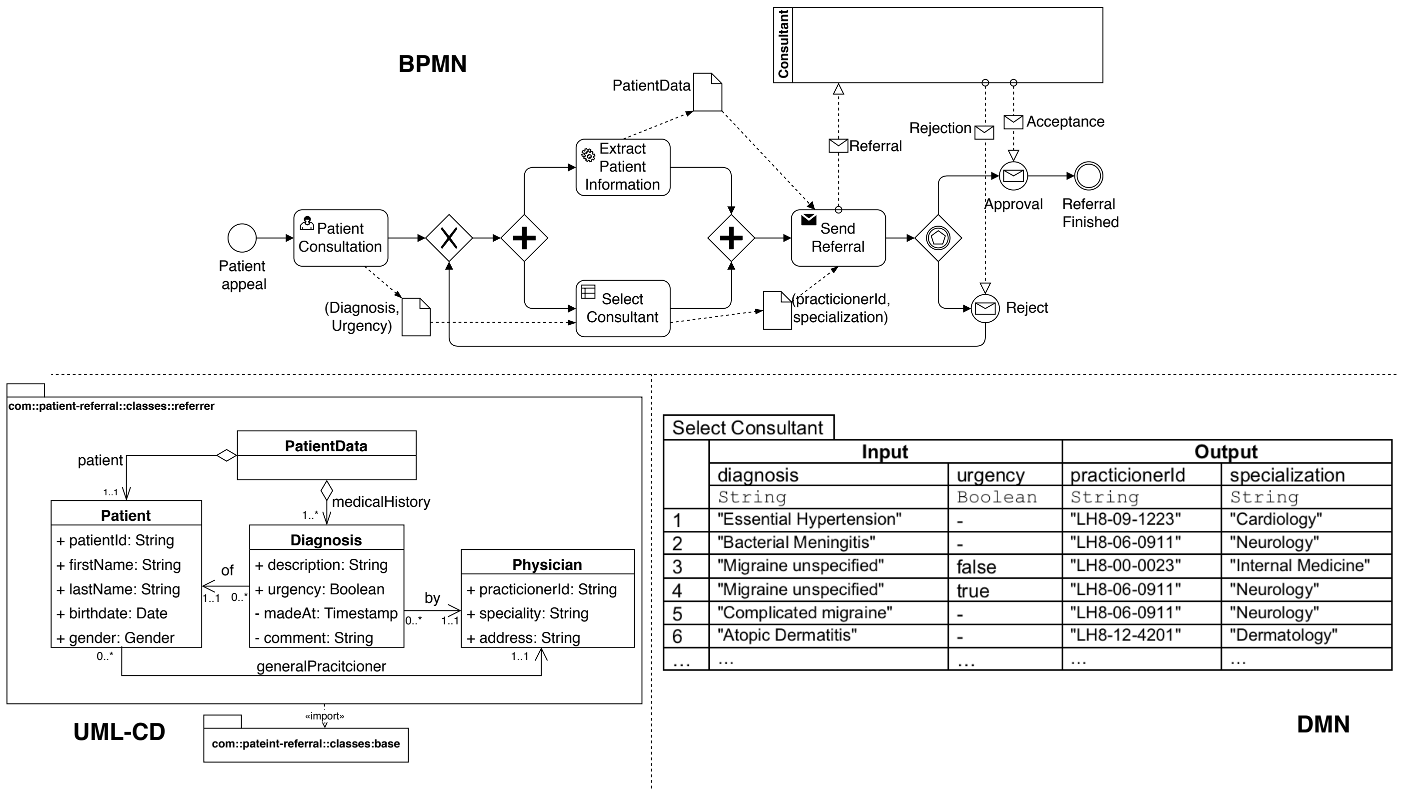


#	Requirement	Mandatory?
1	The system must support creating and editing medication plans.	Yes
2	The system must support checking the availability of drugs.	Yes
3	The system may support to directly order drugs.	No

```java

public class ServiceFacade {

    /**
     *
     * Creates a medication plan for the given patient based on his previous journal history and diagnosis.
     *
     * @implements requirement #1
     *
     */
    public MedicationPlan createMedicationPlan(Patient patient, PatientJournal journal, Diagnosis diagnosis) {

    /**
     * Updates a given medication plan.
     *
     * @implements requirement #1
     */
    public void updateMedicationPlan(MedicationPlan plan) {

    /**
     * Checks how many units of the given drug are still available.
     *
     * @implements requirement #2
     */
    public int checkDrugAvailability(Drug drug) {

    /**
     * Orders a given drug in the given quantity from the central repository.
     *
     * @implements requirement #z
     */
    public void orderDrug(Drug drug, int quantity) {

}
```

```
▼ AllTests [Runner: JUnit 4] (0.000 s)
  ▼ UnitTests (0.000 s)
    ▼ ServiceFacadeTest (0.001 s)
        testCheckDrugAvailability (0.001 s)
        testUpdateMedicationPlan (0.000 s)
        testCreateMedicationPlan (0.000 s)
        testOrderDrug (0.000 s)
```

Figure 1 Scenario 1: Requirements - Code - Tests



Select Consultant

	Input		Output	
	diagnosis	urgency	practicionerId	specialization
	`String`	`Boolean`	`String`	`String`
1	"Essential Hypertension"	-	"LH8-09-1223"	"Cardiology"
2	"Bacterial Meningitis"	-	"LH8-06-0911"	"Neurology"
3	"Migraine unspecified"	false	"LH8-00-0023"	"Internal Medicine"
4	"Migraine unspecified"	true	"LH8-06-0911"	"Neurology"
5	"Complicated migraine"	-	"LH8-06-0911"	"Neurology"
6	"Atopic Dermatitis"	-	"LH8-12-4201"	"Dermatology"
…	…	…	…	…

Figure 2 Scenario 2 - Referral Process: BPMN - UML-CD - DMN

a relation to a test case. Depending on the response to the first change, the desired solution would not naively create yet another test stub but instead link the previously created stub to the new method.

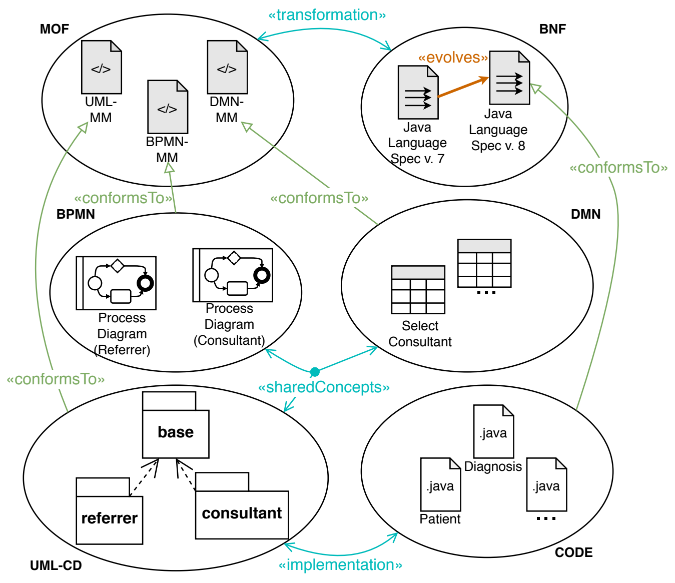


Figure 3 Scenario 2: Model Spaces

The simple scenario above already demonstrates common evolution challenges such as *correctness*, *ambiguity* and *incrementality*. Let us now look at a real life scenario featuring a *patient referral process*. A referral is "the act of sending a patient to another physician for ongoing management of a specific problem with the expectation that the patient will continue seeing the original physician for coordination of total care" (Segen 1992). This scenario comprises both low-level java sources (CODE) and high-level specifications given as process models denoted in *Business Process Model and Notation* (BPMN) (Object Management Group 2014), *data* models denoted as *Unified Modeling Language Class Diagrams* (UML-CD) (Object Management Group 2015a), and *decision* tables given in *Decision Model and Notation* (DMN) (Object Management Group 2019). Fig. 2 shows a selection of the models in this scenario: A BPMN process from the perspective of the *referrer*, a UML package with data type definitions, and a DMN table defining the behavior of the `Select Consultant` activity in the process diagram. The overview of all the models in this scenario is shown in Fig. 3. There is another process model for the perspective of the *consultant* physician, additional UML packages with data type definitions and several more decision tables, one for each business rule activity.

The models in Fig. 3 are grouped into so-called homogeneous *model spaces*. This gives rise to different types of intermodel relationships. In the scenario in Fig. 1, there are relations between elements from disparate models visualized via cyan-colored links. Scenario 2 also comprises this particular type of inter-model relations, which we call *correspondences*: BPMN process, UML class diagrams and DMN tables *share* concepts, e.g. `Diagnosis` in Fig. 2, and UML classes have implementations in CODE. In addition to these correspondences, we have the green *conformance*-arrows in Fig. 3 which are relations between a whole model space and the metamodel of the modeling language of the respective model space. (Favre & NGuyen 2005; Bézivin 2004) The upper half of Fig. 3 shows two model spaces MOF (Object Management Group 2016b) and BNF (programming language grammar definitions) whose members are metamodels (displayed with shaded background) for the models in the other model spaces BPMN, DMN, UML-CD, and CODE.

All these relations are again subject to consistency rules but in this scenario, model repair has to consider even more aspects. The first one is *precedence*, i.e. some models are more "important" than others. For instance, we may consider models in UML-CD to have precedence over implementations in CODE such that the latter can be automatically derived from the former. This transformation (code generation) is defined on the metamodel level as evident from the correspondence between MOF and BNF in Fig. 3. Still, the CODE-artifact should not simply be overwritten upon modifications in UML-CD since the source code can contain intermediate local changes. This again alludes to the aspect of *incrementality*, i.e only repairing the part of a model that is affected. This is a common issue in code generation and *round-trip engineering* (Antkiewicz & Czarnecki 2008). The situation becomes more complicated when neither model has precedence over the other. Note, that the precedence is most likely a partial order rather than a total one, e.g. we neither consider BPMN, DMN nor UML-CD to have precedence over the others. Thus, there is the aspect of *concurrency*: Multiple models can be modified in parallel resulting in conflicting versions that need manual reconciliation. Finally, one must consider different *authorities*, e.g. the metamodels in MOF and BNF are outside of the influence of the patient referral developers and when such external artifacts evolve (see the orange-colored arrow representing the evolution from Java 7 into Java 8), the developers have to adapt their instances but not the other way round.

3. Model Management

Since models are the primary artifacts in MDE, one needs tools supporting the complete life cycle of a model. This concept is known as *model management* (Bernstein 2003; Bézivin et al. 2005; Paige et al. 2009) and it was first proposed in the database domain (Bernstein 2003). It was afterwards adopted in the MDE domain (Favre & NGuyen 2005) resulting in the notion of a *megamodel* (Bézivin et al. 2004), i.e. a model whose elements are in turn models and inter-model relations. In the MDE domain, the *Eclipse Modeling Framework (EMF)* (Steinberg et al. 2008), a meta-modeling and code generation framework quickly became the de-facto technological foundation of the MDE community. EMF alone does not provide much more than a common *exchange format* (Ecore/XMI (Object Management Group 2015b)) and generic model editors. Thus, building on the model management idea, several tool-"ecosystems" evolved around EMF; e.g., the *ATL* (Bézivin et al. 2005) model transformation engine, the *Epsilon* framework (Paige et al. 2009), or the graph-based *eMoflon* (Leblebici et al. 2014).

In the example of the previous section, we motivated *consistency management*, which is a central task in model management (Finkelstein et al. 1992; Nuseibeh et al. 2000; Spanoudakis & Zisman 2001; Sabetzadeh & Easterbrook 2005) comprising consistency verification and consistency restoration. When consistency restoration is applied in an integrated manner on a network of models and inter-relations, one speaks of (co-)*evolution* (Lämmel 2014; Mens et al. 2005; Di Ruscio et al. 2012), i.e. changes on one model trigger changes on other models to maintain global consistency. Literature distinguishes between two cases: *Horizontal* and *vertical* evolution. Horizontal evolution is also known as *update propagation* in the cross-disciplinary research domain bx (Czarnecki et al. 2009)—keeping two related sources of information consistent with each other—while vertical evolution is known as *model migration* (Gruschko et al. 2007; Hebig et al. 2017)—restoring conformance of a model w.r.t. to its metamodel when the latter is changed.

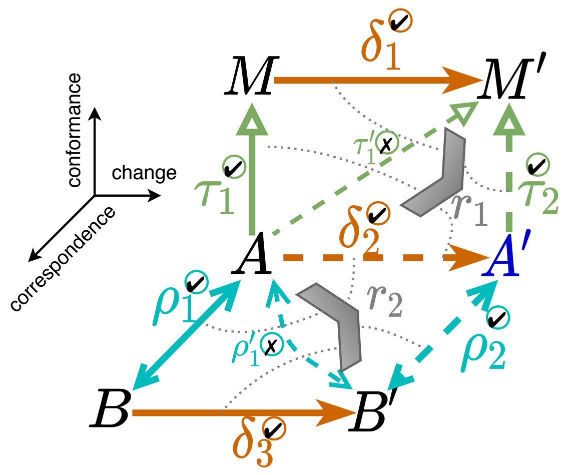


Figure 4 3D Space of Model Management

To provide a high-level overview of model management and evolution, we adopt the conceptual framework for model management developed by Diskin, Maibaum and collaborators in a series of publications, see (Diskin 2009; Diskin et al. 2012, 2013, 2015). The framework organizes model management into a three-dimensional space comprising the dimensions *change*, *conformance* and *correspondence* (see Fig. 4). It comprises models denoted by capital letters ($A, B, M, \ldots$) and inter-model relations denoted as arrows going into respective dimensions (see also Fig. 3). The conformance relation, colored green and denoted by τ, expresses that one model is an instance of (meta-)model, see Sect. 2. The change relation, colored orange and denoted by δ, expresses a model modification, e.g. due to new requirements, regulations, technologies, etc. The *correspondence* relation, colored cyan and denoted by ρ, expresses connections between elements from disparate models, see Fig. 1.

Besides models and relations, there is a notion of *consistency*, which can be thought of as a boolean property (visualized as check- and crossmarks) attached to relations τ, δ and ρ. Consistency on a conformance represents the well-known notion of *intra*-model consistency, i.e. a model adhering to all syntactical and semantical consistency rules of the respective metamodel. Consistency on a correspondence represents *inter*- or *multi*-

model consistency (Diskin et al. 2011). The notion of consistency on changes alone without further associated conformance and correspondence is less common but one may interpret it to express the notion of *authorization* and *validity* of changes. The final ingredient in this framework is the so-called *tiling* operations $r1, r2$ (Anjorin et al. 2019; Diskin 2009), depicted as chevrons. Tiles take model relations as input and produce new models and consistent relations as output. Many model management operations can be modeled via tiling operations (Diskin 2009). In Fig. 4, we displayed the two instances of evolution: The top part shows an evolution of M into M' via change δ_1 such that the instance A does not conform to M' anymore (see inconsistent τ'_1. The tiling operation r_1 expresses model migration. It takes conformance τ_1 and change δ_1 as input and produces the change δ_2 to A such that the resulting A' conforms to M' via τ_2. The bottom part shows an evolution of B into B' via δ_3 such that the original correspondence is violated (see inconsistent ρ'_1). The tiling operation r_2 models update propagation. It takes correspondence ρ_1 (think multi-model comprising A and B) and change δ_3 as input and produces the change δ_2 such that the resulting A' is in correspondence ρ_2 with B'. In the following sections, we will demonstrate how these two tiling operations can be implemented by a more generic operation called *model repair* (Macedo et al. 2017).

4. Model Repair

We begin with an abstract formal definition of model repair based on (Macedo et al. 2017). To clarify the notation: Sets will by convention start with an uppercase letter $(A, B, \dots, M, \dots)$ and functions start with lowercase letters $(f, g, \dots)$. With $f : A \to B$ being a function, we call sets A and B *domain* and *codomain* respectively. Furthermore, there are special sets: $\mathbb{N} = \{0, 1, 2, \dots\}$, i.e. the set of natural numbers, and $\mathbb{B} = \{\top, \bot\}$, i.e. the set of boolean values *true* $(\top)$ and *false* $(\bot)$. The cartesian product of two sets A and B is denoted $A \times B$, an n-ary cartesian product over A is denoted A^n for $n \in \mathbb{N}$ and the powerset of a set A, i.e. the set of all subsets of A, is denoted $\wp A$. Note that every n-ary product $A_1 \times ... \times A_n$ and its subsets (relations) come equipped with a family of n projections denoted $(\pi_1 : A_1 \times \dots \times A_n \to A_1), \dots, (\pi_n : A_1 \times \dots \times A_n \to A_n)$.

Definition 1 (Model Repair)**.** A model repair problem $\mathcal{R}$ is a triple:

$$\mathcal{R} = (MS, CR, rep) \qquad (1)$$

comprising

- A *model space* $MS = (M, E, src, trg)$, i.e. a directed multi-graph (transition system) with a set of *models* M (states) and concrete *edits* E (transitions) where $src, trg : E \to M$ assign to each edit $e \in E$ the model prior $src(e)$ and post $trg(e)$ to the edit. Every model space gives rise to *edit sequences* E^* (think *paths* in MS), i.e. ordered tuples of *updates* or *changes* $u = \langle e_1, \dots, e_k \rangle \in E^*$ with $k \in \mathbb{N}$ obeying the condition

$$\forall e_j, e_{j+1} : trg(e_j) = src(e_{j+1}) \qquad (0 \le j < k) \qquad (2)$$

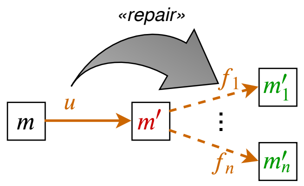


Figure 5 Model Repair

The functions src and trg can be lifted to E^* as follows: $src(u) = src(e_1)$ and $trg(u) = trg(e_k)$ Note, that there is a family of *empty* edit sequence $\langle\rangle_m \in E^*$ for all $m \in M$ $(src(\langle\rangle_m) = m = trg(\langle\rangle_m))$ representing an *idle* update on m.

- A system of *consistency rules* $CR = (P, [\![\_]\!])$ with P being a set of *predicates* a.k.a. *constraints* and $[\![\_]\!] : P \to \wp M$ being a function that assigns to each constraint $p \in P$ the set of *valid* instances $[\![p]\!] \subseteq M$ (semantics). This gives rise to derived functions $check : M \times P \to \mathbb{B}$ $(check(m, p) = \top \Leftrightarrow m \in [\![p]\!])$ and $violations : M \to \wp P$ $(violations(m) := \{p \in P \mid check(m, p) = \bot\})$
- A *repair* function $rep : E^* \to \wp E^*$ that assigns to an input edit sequence u (change) a set $\{f_1, \dots, f_l\}$ $(l \in \mathbb{N})$ of edit sequences (fixes).

A repair problem $\mathcal{R}$ is called *well-behaved* iff it does not introduce new constraint violations, i.e $\forall u \in E^* : \forall f \in rep(u) : violations(trg(f)) \subseteq violations(trg(u))$. It is called *consistency improving* iff the fix produces a "less-inconsistent" result, i.e. $\forall u \in E^* : \forall f \in rep(u) : violations(trg(f)) \setminus violations(trg(u)) \neq \emptyset$, and iff additionally the result shows no inconsistency, i.e. $\forall u \in E^* : \forall f \in rep(u) : violations(trg(f)) = \emptyset$, it is called *fully consistency restoring* (Macedo et al. 2017).

Fig. 5 provides a graphical intuition for Def. 1: Models are depicted as rectangles and edit sequences are depicted as orange arrows. The repair function is shown as a fat shaded arrow. In general, the repair should not produce arbitrary fixes for a given update but strive to produce consistent results (note the coloring of the models: red for inconsistent and green for consistent) — hence the notion of well-behaved, consistency improving and fully consistency restoring. They have been introduced in (Macedo et al. 2017) as the three stages of *correctness*, the core property of model repair.

Furthermore, note that the codomain of the repair function is a powerset. Thus, a repair may produce *no* result (empty set), *exactly one* result (singleton set), or *many* results; see the discussions about ambiguity in Sect. 2. *Ambiguity* and *uncertainty* (Eramo et al. 2015) usually necessitate human interaction, policies, metrics and/or randomness in the repair process. A repair function is said to be *complete* (Hermann et al. 2011) when for any input at least one result is produced, i.e. there are no non-repairable models. Moreover, model management in practice involves repeated invocations of the repair function, i.e. multiple instances of Fig. 5. Results of later repair invocations can depend on the results of previous invocations; see e.g. the Scenario 1 in Sect. 2 where the outcome of the first repair step affects the repair in the second step. Thus, *incrementality* (Egyed 2007; Diskin et al. 2010) is an important issue in model repair. A model repair function may further be composed of other repair functions. *Compositionality* (Foster et al. 2007; Diskin et al. 2019), i.e. preservation of properties from elementary model repair functions under composition is another aspect of model repair that has been a focus of many theoretical studies in this area, such as (Johnson & Rosebrugh 2017, 2016).

Remark 1 (State-Based vs. Delta-Based). Our Def. 1 is considered *delta*-based (Diskin et al. 2010) since *rep* works with changes as inputs and outputs. Alternatively, we could have provided a *state*-based definition of model repair, which takes an inconsistent model as input and produces possible consistent models as output. The state-based notion is popular especially among implementations that are based on model finding and logical solvers (Eramo et al. 2012; Kleiner et al. 2010). However, the delta-based setting is "richer" because it provides full *traceability*. Moreover, the state-based setting is subsumed by Def. 1 when considering MS to be strongly connected, i.e. $E = M \times M, src = \pi_1, trg = \pi_2$ and the existence of a special *empty* model $0 \in M$ such that inputs to *rep* are of the form $(0, m) \in E$ for any $m \in M$.

We can already use the notion of Def. 1 to implement model migration: A metamodel can be interpreted as a system of consistency rules and conformance is given when the instance *satisfies* all rules. Updating the metamodel means changing the consistency rule set. To co-evolve the instance, we apply the model repair operation with respect to the new consistency rule set. Usually, conformance also comprises structural *typing*, which requires additional treatment and we will discuss this further in Sect. 6.6. The other evolution operation in model management—update propagation—cannot be modeled directly by Def. 1 and requires an extension of the theoretical framework, which we discuss in the following section.

5. Generalization: Multi-Model Repair

The *repair* function defined in the previous section cannot directly be applied to our scenarios in Sect. 2. The obvious mismatch is that a multi-model is based on a multi-ary correspondence relation comprising a collection of models $m_1, \ldots, m_n$ ($n \in \mathbb{N}$) while the repair function is based on a single model.

The abstract formal framework proposed by Stevens (Stevens 2008), which is considered the foundation of bx research discipline, extends model repair to a binary setting:

Definition 2 (Pairwise Model Synchronization)**.** A *pairwise model repair (synchronization)* problem is a quintuple

$$\mathcal{S} = (S, T, R, fRep, bRep) \tag{3}$$

comprising

- A *source* model space $S = (M_S, E_S, src^S, trg^S)$.
- A *target* model space $T = (M_T, E_T, src^T, trg^T)$.
- A *correspondence* relation $R \subseteq M_S \times M_T$.
- A *forward repair (propagation)* function $fRep : E_S^* \times M_T \to \wp E_T^*$ such that $\forall u \in E_S^*, t \in M_T : \forall f \in fRep(u, t) : src^T(f) = t$.
- A *backward repair (propagation)* function $bRep : M_S \times E_T^* \to \wp E_S^*$ such that $\forall u \in E_T^*, s \in M_S : \forall f \in bRep(s, u) : src^S(f) = s$.

When comparing Def. 2 to Def. 1, the *local* consistency rules CR are replaced by a *global* correspondence relation R. Predicates and their semantics can be seen as a unary relation on M. Thus, we can derive the function $check_R : M_S \times M_T \to \mathbb{B}$. Furthermore, there are *two* repair functions. The idea is that a local update happening in S or T violates the consistency of a pair $(s, t) \in M_S \times M_T$, which is resolved by calculating a fix for the related model space. *Correctness* and other model repair properties can be seamlessly transferred to pairwise model synchronization. However, there is a new aspect because one now has to work with different *paradigms* (Diskin et al. 2011) due to heterogeneous model spaces.

In general, for $n \geq 2$, the above notion becomes infeasible since it requires forward/backward repair functions for each pairing over the n models and not every n-ary relation can be factored into a binary relation, see (Klare et al. 2019; Stevens 2017). As a result, there has been an increased interest in keeping more than two models consistent, see e.g. (Cleve et al. 2019). One generic alternative approach resulting from these recent research activities is a *reification* of correspondence relations (Diskin et al. 2019), i.e. a global system, which internalizes models and correspondences:

Definition 3 (Multi-Model Synthesis, see Fig. 6)**.** Let $k \in \mathbb{N}$ and $MS_1, \ldots, MS_k$ a set of local model spaces, $CR_1, \ldots, CR_k$ local consistency rules on these model spaces. Furthermore, let $R \subseteq M_1 \times \ldots \times M_n$ an n-ary correspondence relation with $n \in \mathbb{N}$. A *multi-model synthesis* w.r.t. R is a sextuple

$$\mathcal{M}_R = (MS^+, CR^+, glob, proj, rules, comb, deriv) \tag{4}$$

comprising

- A *global model space* $MS^+ = (M^+, E^+, src^+, trg^+)$ comprising all local model spaces, i.e $MS_1 \subseteq MS^+, \ldots, MS_k \subseteq MS^+$.
- *Global consistency rules* $CR^+ = (P^+, [\![\_]\!]^+)$, which encompasses all local consistency rules, i.e. $P_1 \subseteq P^+, \ldots, P_k \subseteq P^+$ and $[\![p]\!]^j \supseteq [\![p]\!]^+$, for all $p \in P_j (1 \leq j \leq k)$.
- A *global view* function $glob : R \to M^+$ that reifies a correspondence relation as a global model.
- A *projection* function $proj : MS^+ \to R$ that restores the underlying correspondence relation of a global model such that:
$$\forall r \in R : proj(glob(r)) = r. \tag{5}$$
- A function *rules* : $R \to P^+$ that translates the underlying semantics of a correspondence relation into a representation as global consistency rules.
- An edit *combination* function $comb : R \times (E^*)^n \to (E^+)^*$ that translates a collections of edits on multi-model components into a global representation such that it holds:
$$\forall r \in R, (u_1, \ldots, u_n) \in (E^*)^n : glob(r) = src^+(comb(r, u_1, \ldots, u_n)). \tag{6}$$

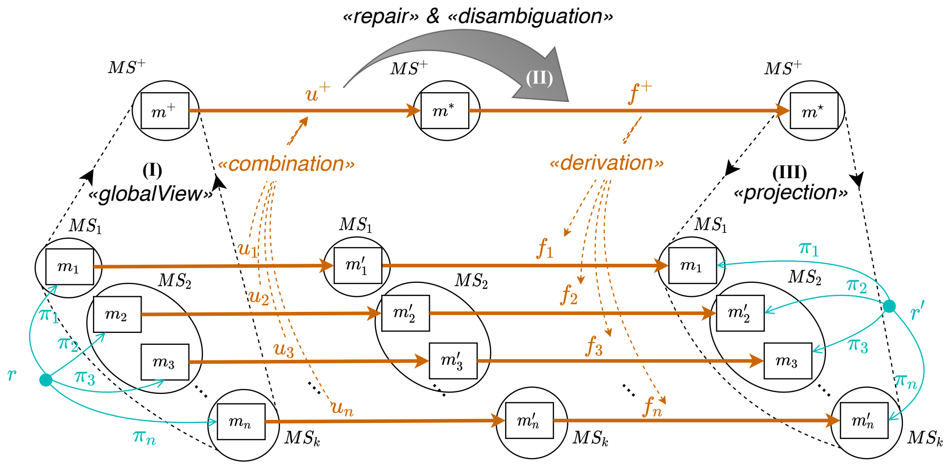


Figure 6 Global and Local Notions of Repair

– A function $deriv : (E^+)^* \to (E^*)^n$ that *derives* a local representation as a family of n updates from a global update such that it holds

$$\forall u^+ \in (E^+)^*, 1 \le i \le n :$$
$$\pi_i(proj(src^+(u^+))) = src^i(\pi_i(deriv(u^+))) \ \wedge \quad (7)$$
$$\pi_i(proj(trg^+(u^+))) = trg^i(\pi_i(deriv(u^+)))$$

To illustrate Def. 3 consider Fig. 6: On the bottom left there is a collection of n models grouped into k model spaces and among them is a correspondence r. The whole collection forms a multi-model. Now, we consider a family $(u_1, \ldots, u_n)$ of updates (some of them probably being idle) on this multi-model. Using *comb*, we construct a global representation u^+ of this family of updates. Thus, we can reuse an existing repair mechanism on MS^+ w.r.t. CR^+ to produce fixes on the global view. Given that this repair produces at least one correct result and there is a disambiguation procedure (e.g. user interaction), we get a fix f^+ resulting in $m^\star$. Finally, we can use *deriv* to derive a family of fixes $(f_1, \ldots, f_n)$ resulting in an updated correspondence (multi-model) r'. This allows us to implement the update propagation via classical model repair leveraging multi-model synthesis.

Synthesis shows well-behaved compositional properties, which were theoretically investigated in (Diskin et al. 2019), and allows reuse of existing research on model repair (Macedo et al. 2017). Still, it is dependent on several technical and conceptual premises. On the technical side, one has to consider that the models are generally based on heterogeneous modeling *formalisms* (Diskin et al. 2011). Thus, there is a hidden requirement for a common metalanguage (Stünkel et al. 2020) to serve as an underlying carrier format. Another technical aspect that is often neglected is *communication*: There are different ways to facilitate information exchange, e.g. file transfer, shared databases, remote procedure calls (RPC), or messaging (Hohpe & Woolf 2012). Model management often assumes that all models are accessible in a single data store. On the conceptual side one has to consider that there may be *authority*, *privacy* and *concurrency* issues, which are hidden by the construct in Def. 3. In Sect. 2, we shortly discussed *authority* and *precedence* of some model (spaces) over the others. Thus, repair has to take into account that some models (elements) are rather allowed to change than others. Privacy restricts the degree of information that can be shared globally, which aggravates consistency verification and model repair. Finally, the members of a multi-model may be subject to concurrent updates where each update is consistent in itself but in parallel it results in an inconsistent state. Thus, multi-model synchronization (Antkiewicz & Czarnecki 2008) shows similarities to *optimistic replication* (Saito & Shapiro 2005) in distributed systems, which has only been investigated by a minor number of publications (Xiong et al. 2013; Hermann et al. 2012) while the majority of publications pertains to a *serial* (Diskin et al. 2019; Hermann et al. 2011) setting where changes only happen on one model at a time. We will investigate these details in greater detail in Sect. 6.5.

6. State of the Art

In the previous sections we have introduced the core concepts of model management, i.e. *models*, the three relations *change*, *conformance* and *correspondence*, as well as the *repair* operation, which is central for implementing model (co-)evolution. The remainder of the paper will provide an overview of existing work and approaches. There are several existing surveys in different sub-domains, i.e. model repair (Macedo et al. 2017), update propagation (bx) (Anjorin et al. 2019; Hidaka et al. 2016; Diskin et al. 2016), multi-view modeling (Cicchetti et al. 2019; Bruneliere et al. 2019; Knapp & Mossakowski 2018), and model migration (Hebig et al. 2017). Naturally, there are several overlaps between these surveys both w.r.t. references and feature models, which are used to classify findings.

To provide a comprehensive overview of the state of the art, we develop a feature model that accommodates for the findings in the above surveys and is aligned with the model management framework depicted in Fig. 4. The top layer of our feature model is depicted in Fig. 7, where clouds are used to relate the topmost abstract features to the respective model management

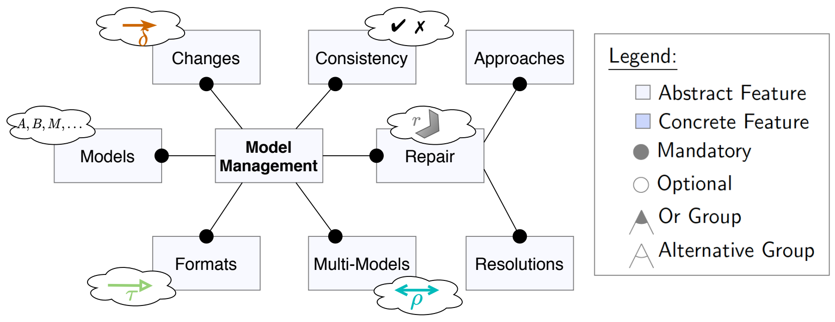


Figure 7 Feature Model Overview

concept (the legend applies for all the following figures). In the following subsections, we will provide a detailed explanation of each abstract feature.

6.1. Survey Method

Our literature study is not a systematic literature review or systematic mapping study (secondary study) as described e.g. in (Kitchenham et al. 2007) but instead a meta-analysis (tertiary study) of the existing surveys mentioned above. We started by reading those surveys, collecting and reviewing the contained references, and finally comparing the feature models from each survey. Afterward, we identified the common features and developed our comprehensive feature model in alignment with the model management concepts from Sect. 3, which is based on the model repair feature model in (Macedo et al. 2017).

6.2. Models Feature

Models are the elementary artifacts in model management and thus a core feature (it was termed *Domains* in (Macedo et al. 2017)) which mainly distinguishes *technological space* (concrete syntax (Cicchetti et al. 2019)) and *formal* (abstract syntax) representation (see Fig. 8).

Technological Space Model management in the database domain, e.g. schema migrations and view updatability (Roddick 1992; Bancilhon & Spyratos 1981), is based on *SQL*. Some approaches are specific for a certain *programming language (PL)*. Most common choices are *JVM* languages (Buchmann 2019; Leblebici et al. 2014), the *.NET* framework (Hinkel & Burger 2019), *Haskell* (Ko et al. 2016) and *Prolog* (Pinna Puissant et al. 2012). Arguably, the majority of tools uses *XML*, where we can further classify *free form* or *custom schema* approaches (Nentwich et al. 2003) and specific *XML schemas* (Reder & Egyed 2012) such as XMI, which is the OMG standard for encoding UML and MOF models. The popular *EMF* framework (Steinberg et al. 2008) is based on a simplified MOF metamodel and utilizes XMI to encode metamodels and models. Remaining technologies, such as e.g. the graph language DOT (Hidaka et al. 2011), constitute only a small share grouped as *Other*.

Formal Representation Arguably, the most simple modeling formalism is to consider a model as a *set* of its elements (Foster et al. 2007). Research in the database domain (Bancilhon & Spyratos 1981) naturally utilizes the *relational model* (Codd 1970) as a formal underpinning. A different, but formally equivalent, approach is to consider a model as a set of facts i.e. *logical* sentences. This is the standard formalism for tools based on solvers (Macedo & Cunha 2016; Cicchetti et al. 2011). In functional programming (Ko et al. 2016), the metamodel is defined via abstract datatypes, which abstractly represent *trees*. Strictly more general than trees are *graphs* (Hidaka et al. 2011), which allow for cyclic relationships between elements and are considered a standard means for depicting structured information in Computer Science. The *object oriented (OO)* formalism (Reder & Egyed 2010; D. Kolovos et al. 2008) combines aspects of both graphs and relational formalisms and adds additional features such as inheritance and constraints. The most abstract formalism (Hinkel & Burger 2019; Boronat & Meseguer 2010) is given through concepts borrowed from *Category Theory* (Barr & Wells 1990), which generalize all of the above, see e.g. (Diskin & Wolter 2007; Rutle et al. 2012)

6.3. Changes Feature

There is no evolution without *changes*, a core feature shown in detail in Fig. 9. Models and changes together are featured as model spaces in Def. 1 .

Representation and Definition In Remark 1 in Sect. 4, we discussed the two paradigms for representing changes: Either a change is identified with its result (Bancilhon & Spyratos 1981; Foster et al. 2007) (state-based) or a change is an entity on its own right (Bergmann et al. 2012; Diskin et al. 2010) (delta-based). A delta can be represented in a *structural* or *operational* way (Anjorin et al. 2019). *Structural* deltas (Diskin et al. 2010; Hermann et al. 2011) are traceability links depicting which elements are added, which are deleted, and which remain unchanged. *Operational* deltas (Hofmann et al. 2012) are the actual method invocations being executed. Another feature aspect is the distinction whether the set of edits is *undefined*[1] (when any kind of modification is a valid change), *fixed* (Reder & Egyed 2012) or *customizable* (Macedo & Cunha 2016) by composition of elementary built-in operation. Note that a state-based change representation does not always imply an undefined edit set: See e.g. (Macedo & Cunha 2016), which presents changes in a state-based manner but allows the definition of custom changes to influence the model finding process. Another

[1] Strictly speaking it is not undefined: In this case it holds $E = M \times M$.

approach is to consider changes being implicitly derived from the respective metamodel (Kehrer et al. 2016).

Types and Recording An approach may optionally consider different *types* of changes to be possible/allowed. We distinguish between *atomic* (indivisible) and *complex* (compositions of atomic) changes. The majority of model management frameworks supports the elementary operations: *Insert*, *Update* and *Delete*. Note, that it is important to consider *Rename* as a separate operation even though it could be modeled as a delete followed by an insert, since it can cause rather different side effects, especially in model migration (Hebig et al. 2017). Some researchers also consider more elaborate atomic changes such as *Move*, *Copy* and *Merge* (Hebig et al. 2017). Most complex changes are represented as a *sequence* of atomic changes. However, there are approaches that also consider changes happening in *parallel* (Xiong et al. 2013) or even *contingency plans* (Pinna Puissant et al. 2015), which account for different possibilities.

Changes are *recorded* either *offline* (statically presented to the tool) or *online*, i.e. the tool actively records the changes itself. This can happen *intrusively* (i.e. the user has to use a special interface to record changes) or *non-intrusively* (i.e. the user remains unaware). The advantage of online recording is its ability to immediately represent the change. Offline change recording may provide insufficient information, e.g. only providing the new model state. In this case, *Change Identification* (Alanen & Porres 2003; D. S. Kolovos et al. 2009; Rivera & Vallecillo 2008) is necessary to identify the type of the change and calculate the change representation. Change Identification is a sophisticated problem on its own, a good starting point for further reference is found in (Hebig et al. 2017).

Meta-Information The most common example of change meta-information is some sort of version *history*, e.g. the *previous* state of the model before the change. A delta-based change representation implies that this information is always available. Additionally, the model management tool may have access to the *complete* version history of a model. Furthermore, each change can provide *environment* information, e.g. user credentials, the date and time of when the change happened, a reference to external documents etc.

6.4. Consistency Feature

Fig. 10 depicts the *consistency* feature. Abstractly, consistency is given by a set of rules, see **CR-1** in Sect. 2 and *CR* in Def. 1. Consistency rules are generally attached to conformance relations (intra-model consistency) or correspondence relations (inter-model consistency).

Definition The set of consistency rules is either *builtin* (Reder & Egyed 2012), *customizable* via *combinator libraries* (Ko et al. 2016; Foster et al. 2007), *grammars* (Hermann et al. 2011), *logical theories* (Eramo et al. 2012; Macedo & Cunha 2016), and/or defined *externally* (Pinna Puissant et al. 2015; Nuseibeh et al. 2000). Combinator libraries are primarily utilized in functional programming based synchronization tools, where the notion of consistency is implicitly derived from the semantics of a set of builtin predicates and composition operators. In *Grammar*-based approaches such as algebraic graph grammars (AGG) (Ehrig et al. 2006; Biermann et al. 2008) and triple graph grammars (TGG) (Schürr 1994; Hermann et al. 2011; Leblebici et al. 2014) consistency arises as the set of (multi-)models that can be derived through applications of grammar rules. The implementation of the consistency check is thus based on pattern matching (Macedo et al. 2017). Formally, graph grammars are equally expressive as variants of first-order logic (FOL) (Courcelle 1997; Habel & Pennemann 2009). A common formulation of consistency is by utilizing is a *logical theory* such as FOL (Huth & Ryan 2004). In MDE, the *Object Constraint Language (OCL)* (Object Management Group 2012) is a widely used formal language to define domain-specific consistency rules that cannot be expressed by means of the modeling language. Externally defined consistency rules are outside of the reach of the model management tool, i.e. they rely on external tools or user interaction.

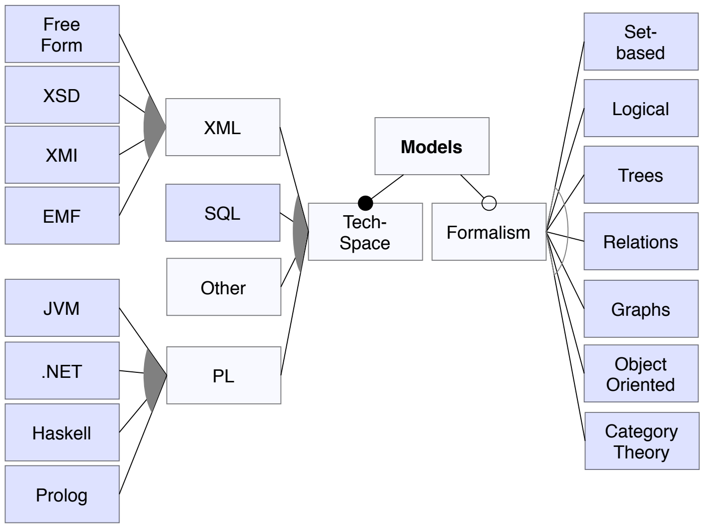


Figure 8 *Models* Feature

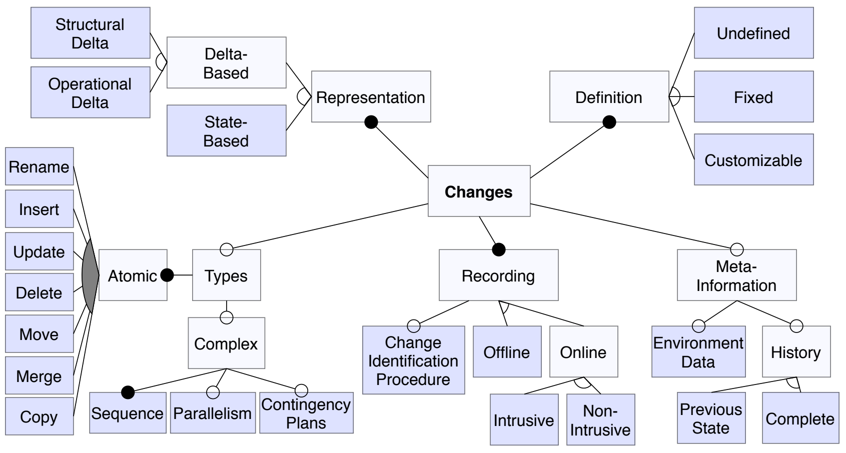


Figure 9 *Changes* Feature

Checking A consistency check may be performed on *on-demand* (Leblebici et al. 2014; Eramo et al. 2012) (i.e. the user explicitly invokes a check function) or *event-based* (Hinkel & Burger 2019; Macedo & Cunha 2016). The check is performed *automatically* or *manually*, e.g. if definition of the consistency rules is external. The information contained in the consistency *report* differs from tool to tool. The plainest type is a *boolean* (Ko et al. 2016) saying if the model is consistent or not. More instructive than a boolean is a *number* telling how many constraints are violated. The most insightful report additionally associates violated consistency rules with the *elements* (Reder & Egyed 2012) that are violating it. Another option is to directly point to the *goal* for the subsequent repair process (Pinna Puissant et al. 2015).

Inconsistency Levels and Meta-information One may further consider different levels of consistency rules. An example is a distinction between proper *constraints* (mandatory) and *critiques* (optional) (D. Kolovos et al. 2008). The number of such levels is either *fixed* by the framework or *customizable*.

Finally, consistency rule can be augmented with different *meta-information*. One type of such meta-information is the *scope* (arity) (Diskin & Wolter 2007; Rutle et al. 2009) of the consistency rule, i.e. the elements affected by a certain rule. This can be used to implement efficient checking via localization (König & Diskin 2017). In rule-based repair approaches, there is a tight relationship between the definition and consistency rules, see Sect. 6.7. This means that a consistency rule already provides information about how it can be resolved in case of inconsistency. We name this principle *repair hints* and they can be quite blatant (Ko et al. 2016; Hinkel & Burger 2019; Samimi-Dehkordi et al. 2018) (explicitly telling how to fix the inconsistency) or rather subtle as in the TGG framework (Hermann et al. 2011). A notable example of such subtle hints are *constructive axioms*, i.e. consistency rules defined as a *horn clause* (e.g. grammar rules), which can be directly translated into a repair rule (Cohn 1965).

6.5. Multi-Models Feature

In general, model management also implies multi-modeling, i.e. one has to deal with a collection of models and relations between them. Multi-Models are a central research object in megamodeling (Favre & NGuyen 2005; Stevens 2020), multi-viewpoint modeling (Cicchetti et al. 2019; Bruneliere et al. 2019) and model synchronization (bx) (Anjorin et al. 2019).

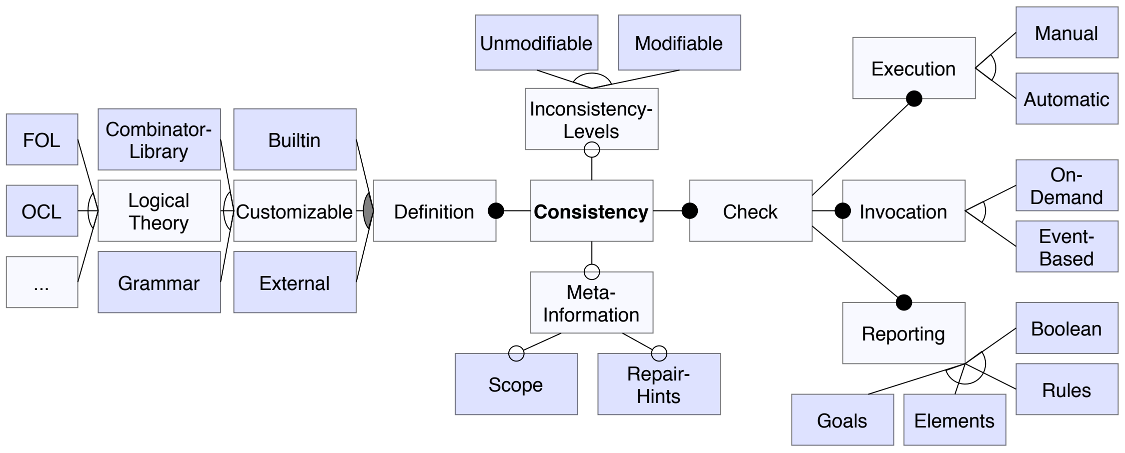


Figure 10 *Consistency* Feature

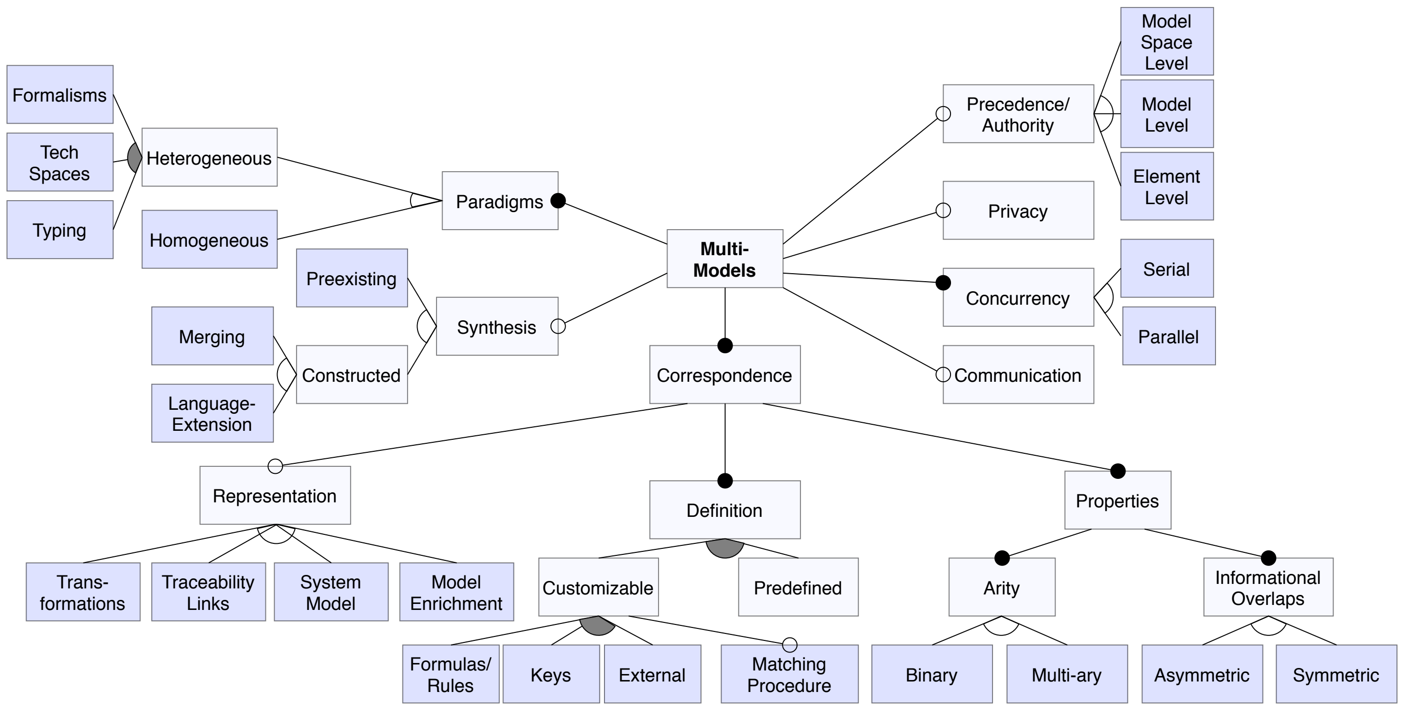


Figure 11 *Multi-Models* Feature

Naturally, they form one of the core features, which is detailed in Fig. 11.

Correspondences The most important sub-feature of multi-models is the correspondence relation, which sets a collection of models in relation. The majority of approaches considers correspondences to be structurally *represented* (Bernstein 2003; Aizenbud-Reshef et al. 2006; Feldmann et al. 2019; Klare & Gleitze 2019; Stünkel et al. 2018; Schürr 1994) in some way but it is also possible to consider correspondences as a black box that is handled outside of the model management framework (Stevens 2020). Based on the findings in (Knapp & Mossakowski 2018; Klare & Gleitze 2019; de Lara et al. 2018) we identify four primary ways of expressing correspondences. Multi-viewpoint modeling (Atkinson et al. 2010; ISO/IEC JTC 1/SC 7 Software and systems engineering 2011) distinguishes between projective and synthetic view modeling, where projective means that there exists a single underlying *system model* and every view model is a projection from parts of this model. Thus, such system models establish correspondences. In the synthetic approach there is no such pre-existing system model and correspondences have to specified differently. One of the most common approaches is based on *traceability links* (Feldmann et al. 2019; Samimi-Dehkordi et al. 2018; Aizenbud-Reshef et al. 2006) also known under the terms: mappings (Bernstein 2003), cross-references (de Lara et al. 2018), commonalities (Klare & Gleitze 2019), or correspondences (Stünkel et al. 2018; Schürr 1994). It means to augment models with an external storage of links, which relate elements from different models. These links can have different semantical interpretations (e.g. similarity, generalization, aggregation, dependency etc.) and may have their own type system (Feldmann et al. 2019; El Hamlaoui et al. 2018). Most programming-based model synchronization approaches (Ko et al. 2016; Foster et al. 2007; Hinkel & Burger 2019) chose a different approach that is called *heterogeneous transformations* in (Knapp & Mossakowski 2018). Model correspondences are not explicitly reified in a system model or as traceability links but implicitly via transformation functions between each pair of models, which translate one model in another, see Def. 2. The fourth and final representation, we call for *model enrichment* (Engels et al. 2000; Knapp & Mossakowski 2018). It is based on augmenting models and/or their metamodels (de Lara et al. 2018) locally with elements from other models.

The correspondences itself may either be *predefined* (a classic example is the UML metamodel itself (Reder & Egyed 2012)), *customizable* or a mix of both. For customization means, we distinguish between *external*, *formulas*, and *keys*. External correspondence definitions are given to the model management tool from the outside (Samimi-Dehkordi et al. 2018), while formulas provide a logic based interface to flexibly define correspondences. An example is the QVT-r language (Object Management Group 2016a; Macedo & Cunha 2016), where developers can define correspondences with the expressive power of OCL. Keys are a less-expressive but are a very efficient variant, where elements are set in relation based on the value of a certain property (often their `name`). It is often used in combination with a system model based correspondence representation. To retrieve a correspondence representation w.r.t. a correspondence definition, a *model matching procedure* (Ehrig et al. 2008; Barbosa et al. 2010; Brunet et al. 2006) may be required; see *change identification* in Sect. 6.3. This can be a complicated endeavor as it is an NP-complete problem (Rubin & Chechik 2013) and often requires human interaction. Furthermore, since correspondences involve multiple models this may require multiple stakeholders to agree. An interesting approach to this problem uses collaborative decision making is found in (Bennani et al. 2019).

Characteristic *properties* of *correspondences* are *arity* and *informational overlaps*. Traditionally bx research has focused on *binary* correspondences (Foster et al. 2007; Stevens 2008), but in practice multi-ary relations are common too, which sparked increased interest in the scientific community on this topic (Cleve et al. 2019; Diskin et al. 2018; Stevens 2020). One possibility is to model multi-ary correspondences through a network of binary ones but there are multi-ary relations that cannot be factored this way (Stevens 2017; Klare et al. 2019). Informational overlaps were thoroughly investigated in (Diskin et al. 2016) which used the terminology *symmetric* vs. *asymmetric*, which in turn stems from the lens framework (Johnson & Rosebrugh 2017). An asymmetric lens describes a situation where all information in one (view) model can completely be derived from another (source) model, whereas in symmetric lens each model contains information which is not also present in the respective other.

Paradigms and Synthesis One has to consider that the members of a multi-model stem from different model spaces and thus are based on heterogeneous modeling paradigms, i.e. different *technologies* and/or *formalisms*. Early model management approaches (Sabetzadeh & Easterbrook 2005) required a *homogeneous* setting, i.e. all models are members of the same model space. A middle ground treats models heterogeneously typed (Diskin et al. 2011), i.e. one can encode them as artifacts conforming to different metamodels but using a common meta-language, e.g. a standard exchange format such as Ecore-metamodels or a universal formalism such as e.g. (attributed) type graphs.

Another aspect is whether the model management environment considers an explicit *synthesis*, see Def. 3, i.e. a way of representing a whole multi-model as a single artifact. Note that in case of projective multi-viewpoint modeling, this artifact is already present. In the synthetic case, one may construct it using one of the two following alternatives that have been identified in the literature: *Merging* (Pottinger & Bernstein 2003; Sabetzadeh & Easterbrook 2005; Brunet et al. 2006) and *Language Extension* (Stünkel et al. 2020; Feldmann et al. 2019; Rabbi et al. 2014). Intuitively, merging means to collect the elements from all models into a new model wherein corresponding elements are identified. This construction was fully formalized in a series of papers (Diskin et al. 2011; König & Diskin 2016, 2017) based on the categorical concept of a *colimit*. A difficulty with model merging is that different multi-models may result in the same merge, which leads to problem if one wants to restore the multi-model representation from a merge. The language extension approach was formally investigated under the name *comprehensive systems* in (Stünkel et al. 2020). Here, elements from all models are collected like in the merge approach but instead of identifying elements, the correspondences are *internalized* as structural links by means of the representation language, which may require a linguistic extension (Atkinson & Kühne 2001) of the modeling language to express the structural correspondences, see e.g. (Rabbi et al. 2014). This approach is heavily used in practice, however, in a formally less rigorous way, see e.g. (Feldmann et al. 2019).

Authority, Privacy, Communication and Concurrency Scenario 2 in Sect. 2 showed that multi-modeling involves a notion of *precedence/authority*. It was thoroughly investigated in (Diskin et al. 2016) where it was termed *organizational symmetries*, i.e. change propagation is only allowed in certain directions. The easiest variant defines precedence on the *level of model spaces* (Stevens 2018) but we may also have to consider a precedence hierarchy among models inside a model space. The most fine-grained but also most complicated notion is to define precedence on the *level of model elements*.

The importance of *Privacy* (Johnson & Stevens 2018) has been identified in bx recently and it was pointed out that it so far has received less attention in the literature. However, in practice it plays a major role in many scenarios with strict legal privacy requirements, e.g. the health care sector. Concretely, this means that not all information contained in a model is actually accessible for consistency checking and/or repair. It necessitates additional filtering/obfuscation and increases the degree of manual activities due to approval processes.

Technical *communication* issues among multiple systems (=models) (Hohpe & Woolf 2012) are traditionally not investigated in the model management literature since most works abstract away from these technical aspects. These problems belong to the research field of *distributed computing* (Tanenbaum & Steen 2007) but may be necessary to take into consideration when working with models, which are only accessible behind web service interface

Finally, *concurrency* is an important and challenging aspect, i.e. changes can happen to multiple models and approaches may require them to be *serialized* (Diskin et al. 2019) (on at a time) or allowed to happen in *parallel* (Xiong et al. 2013; Hermann et al. 2012), see complex change types in Sect. 6.3. Authority can play an important role in coordinating conflicts, which however can lead to overwritten changes and information loss.

6.6. Formats Feature

Every model has to conform to a specific format, otherwise it would not be possible to process them electronically. The respective core feature is detailed in Fig. 12.

Conformance Conformance is the relation between a model (the instance) and its metamodel (format). It is generally expressed via rules (Paige et al. 2016) also called *constraints* in this context to ensure that the instance is semantically valid, see Sect. 6.4. In addition, most notions of conformance comprise *typing*, i.e. every instance-element is assigned to an an element (concept) from the format. This is sometimes referred to as an abstract syntax representation (Cicchetti et al. 2019). Furthermore, a format may already come equipped with a *concrete syntax*, which can be a textual notation or a graphical visualization. Thus, conformance also refers to this aspect.

Representation Since models are represented using different technological spaces and formalisms, formats come in different shapes as well. Hence, our usage of the neutral term format

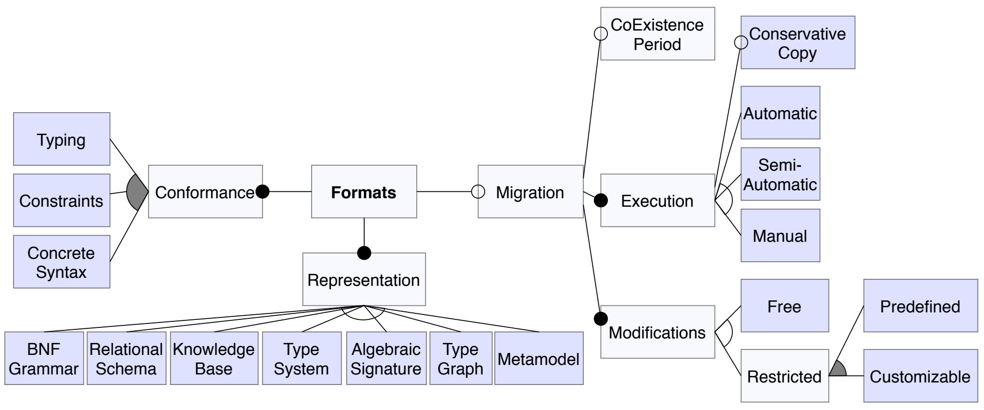


Figure 12 *Format* Dimension

instead of common terms such as metamodel, type or schema to avoid confusion with the concrete characteristics of the *representation* feature. In the technical space of programming languages, the format is expressed as a *BNF grammar* (Vermolen & Visser 2008). In the database world, *relational schemas* are used (Bancilhon & Spyratos 1981). When using a logical representation, the format may be represented as a *knowledge base* (Pinna Puissant et al. 2015), i.e. collection of facts, and instances are those sentences that can be derived from the knowledge base. *Type systems* are common for approaches based on functional programming (Ko et al. 2016). *Algebraic signatures* are yet another formal format (Boronat et al. 2009). *Type graphs* are used as the format representation in graph based formalisms (Biermann et al. 2008), i.e. a distinguished graph is selected as the type and all graphs possessing a structure-preserving mapping into the type graph are valid instances. Finally, many works in MDE work with *metamodels*, which formally combine aspects of a type graph with (OCL) rules. An important feature is that a metamodel is a model itself, which conforms to yet another metamodel thus allowing concepts such as deep metamodeling and multi-level modeling (Kühne 2006).

Migration Some model management approaches may support *migration*, i.e. allowing the format to be changed as well (Hebig et al. 2017; Rose, Herrmannsdoerfer, et al. 2014; Rose, Kolovos, et al. 2014). Here, we can first distinguish between the types of *modifications* that are possible on the format. If we can apply the same changes on formats as we do on instances, we will call them *free* otherwise *restricted*. These restrictions may either be *predefined* or *customizable*, e.g. the developer can specify allowed metamodel modifications (and how to react on them) beforehand (Mantz et al. 2015). Next, there are different ways of how the migration is *executed*. To the extremes, it may either be a complete *manual* or *automatic* process. Most commonly it is a combination of both, where the first step of the migration can be executed automatically (e.g. removing all elements typed over deleted elements) followed by a second step, which requires human interaction and/or supervision. In (Rose, Kolovos, et al. 2014), the authors identified a concept called *conservative copy*, i.e. a part of the current instance that is unaffected by changes happening on the format. In this case, the migration can be executed by first taking a conservative copy and then invoking multiple rounds with model repair on the invalid parts to eventually yield a valid instance. Finally, migration may involve a *period of co-existence* between the old and new version of the format. This is especially common for databases, which are used by many client systems that need to be granted a transition period towards the new format (Ambler & Sadalage 2006).

6.7. Repair - Approaches Feature

The most important part of the repair approaches feature dimension, depicted in Fig. 13, is the abstract sub-feature *Implementation*: The vast-majority of approaches can be classified into two main paradigms: *Rule-based* and *Search-based*.

Search-based Solutions The generic search-based approach is intuitively described as letting the procedure figuring out a solution by "trial-and-error". A repair problem is easily conceived as a search problem: The *model space* represents a transition system by definition (see Sect. 4). It comprises a state space given by the set of all models M. The transitions are given by the set of all edits E. The start state is given by the input $(trg(u))$ and the consistency rules define *goal states*, i.e. those models $m \in M$ which are considered consistent $(violations(m) = \emptyset)$.

The strength of this approach is its domain independence and it can easily be adapted to new or changed scenarios. The weakness and thus the biggest challenge of this approach is its computational complexity. Already for a small input, the naive *atomic search* (Silva et al. 2010; Kessentini et al. 2019) quickly runs out of time and/or memory. It is well known, that so called *heuristics* can improve the efficiency of search algorithms to make complex problems tractable (Pearl 1981). However, finding a suitable heuristic function remains tricky. One may think of generic heuristic function such as the number of violations, however, domain specific knowledge tends to provide even more effective heuristics.

A common variation of the naive atomic search is to consider the problem as a *constraint satisfaction problem (CSP) problem*: In CSP, search space states are not atomic but have an internal structure. This structure is given by a set of *variables*, where each variable can take one value from a given *domain* and every variable is subject to one or more *constraints*. This factored

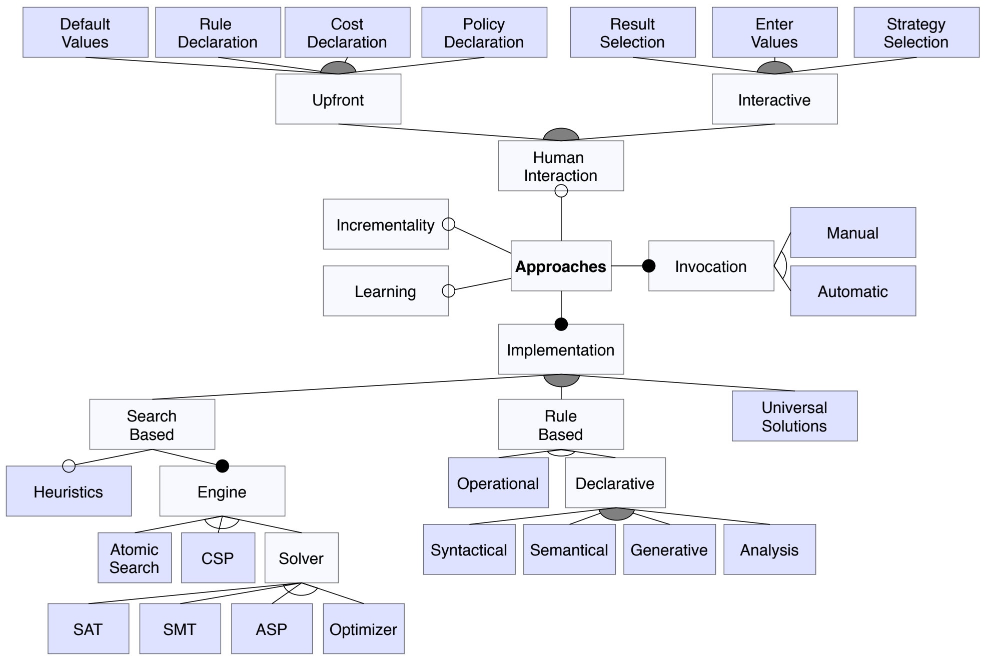


Figure 13 *Repair - Approaches* Feature

presentation allows for a much more effective search because searching means to vary the value of those variables, which are affected by a constraint violation.

Yet another approach is *Satisfaction (SAT)* solving. It actually represents a special case of a CSP where all variables are boolean and all constraints must be formulated as *propositions*. The so-called SAT-solvers represent their own research domain, which has produced remarkable results and performance improvements that can be exploited for implementing model repair. The translation of a whole problem domain together with its behavior and constraints into representation solely consisting of boolean variables and propositions quickly leads to a proliferation of variables and propositions. Thus, a more convenient and high-level way is to use a *Satisfaction Modulo Theories (SMT)* solver, which offers a more abstract interface by providing a set of built-in theories, e.g. arithmetics on integers, manipulations of character-strings, etc. These theories have their own highly optimized translation into the underlying SAT-presentation. An example of an SMT-solver that is often used in the context of object-oriented modeling is *Alloy* (Jackson 2016), which offers a built-in relational theory that in turn resembles object-oriented design and notation. SMT-solvers are a popular choice for model repair and used in a wide number of approaches, e.g. (Kleiner et al. 2010; Straeten et al. 2011; Macedo & Cunha 2016).

The technique used in SAT and SMT solving is called *model checking*, i.e. enumerating all possible models until a solution is found. Instead of using model checking, one can alternatively use syntactical reasoning: The dynamics (possible changes) of the domain are encoded in logical statements and the repair is formulated as a query asking whether a consistent model state can be reached by a sequence of changes. When the query can be fulfilled in the present knowledge base by syntactical reasoning, a repair is found. Implementations of this approach are *Logic Programming (LP)*, *Constraint Logic Programming (CLP)* and *Answer Set Programming (ASP)* (Pinna Puissant et al. 2015; Eramo et al. 2012; Cicchetti et al. 2011) . Due to its nature there are certain restrictions on the type of logical sentences that can be used, e.g. they must be quantifier free, do not contain negation (in case of LP), etc. Another solver-based approach is to formulate the problem as an optimization problem and to use an appropriate algorithm to find the result (Leblebici et al. 2017).

Rule-based Solutions Rule-based solutions explicitly tell the program *how* to fix an inconsistency. These instructions are given as *rules* in the form: IF *condition* THEN *action*. A *condition* represents a specific constraint violation and *action* is a sequence of edits fixing this inconsistency. Thus, in rule-based approaches the definition of consistency and repair rules is often tightly connected. It heavily depends on the domain expert to define the *right* set of rules. Thus, rule-based solutions are not universal and cannot easily adapt to varying scenarios. The strength of this approach, however, is its efficiency: When the rule, which shall be applied, is found (match), the repair itself happens in constant time. Finding the *right* rule, however, can be challenging, i.e. finding rule-matches becomes a search-problem of its own (Gomes et al. 2014).

Rule-based solutions can be classified into *operational* (Samimi-Dehkordi et al. 2018; Nentwich et al. 2003; Xiong et al. 2009) and *declarative* (Hermann et al. 2011; Reder & Egyed 2012). Operational rules are procedures or functions written in a programming language. In general, operational rules provide no guarantee that they actually lead to the desired result and it is up to the user to define the rule correctly. Thus, researchers came up with *declarative* rules (Mens & Van Der Straeten 2007), which actually can be statically analyzed.

The most popular declarative rule-based framework is algebraic graph transformation (Ehrig et al. 2006), which offers powerful means to statically analyze concurrency, confluence, conflicting and termination properties of the rule set. An example of a rule-based approach combining graph transformation and user interaction is found in (Nassar et al. 2017, 2018). Another feature of the declarative approach is that it is able to *generate* (Reder & Egyed 2012; Ehrig et al. 2007) rules automatically, which significantly reduces the manual effort. It is usually based on grammars (i.e. abstract rule sets), which can be classified into *syntactic* (Reder & Egyed 2012) and *semantic* (Schürr 1994; Hermann et al. 2011) categories. The syntactic category exploits the fact that (modeling) languages are generally defined in terms of a grammar, which can be used to derive potential changes. The semantic category additionally requires the consistency rules to be defined in terms of a grammar as well, see Sect. 6.4.

Universal Solutions A third category of implementations is identified in BX domain. Cicchetti et. al. (Cicchetti et al. 2019) termed this approach *proxy*-based but we will call it *universal solutions* due to the underlying concept from category theory (Barr & Wells 1990). Intuitively it can be described by replacing a part of a (view) model with the content from another (source) model and leaving the rest unchanged. The replaced part is determined by a function, the so-called view definition. This principle a.k.a. *constant complement* was first identified in databases (Bancilhon & Spyratos 1981) and later inherited by the programming language community, which formalized it as an algebraic design pattern known as *lens* (Foster et al. 2007). The framework was developed further by category theorists (Johnson et al. 2010; Johnson & Rosebrugh 2007) who identified further applications of universal properties, which practically imply removing specific elements or freely adding missing elements. Universal solutions play an important role in a class of programing based multi-model repair tools comprising *BiGUL* (Ko et al. 2016), *Boomerang*, *GRoundTram* (Hidaka et al. 2011) (Foster et al. 2007) and *NMF* (Hinkel & Burger 2019).

Human Interaction Several researchers (Reder & Egyed 2012; Nassar et al. 2017; Ludovico et al. 2020) have argued that the user should play a leading role in the model repair process. We distinguish several ways of human interaction, coarsely grouped into *upfront* (performed before the repair invocation) and *interactive* (performed during the repair) measures. A classical example of an upfront measure is the definition *default values* to be used when new elements are created during the repair. An upfront measure with less obvious implications are *cost definitions* (Macedo & Cunha 2016), which associate a cost with edit operations thus influencing the repair results because the tool will try to minimize these costs. Another sophisticated tool is *policies*. A policy can be compared to consistency rules: they declaratively require some properties to hold, e.g. avoid deletion in M1 in Fig. 1. The difference to consistency rules is that they are invoked later in the process to disambiguate between multiple valid choices. The predestined example for an interactive measure is *result selection*: The tool calculates all possibilities resulting in a consistent model and presents it to the user, who has to pick his preferred choice. The *interactive* equivalent of default values is to request the user to *enter values* for missing attribute values on the way. *Strategy selection* is similar to result selection with the difference that the outcome is unclear, i.e. the tool presents the user with possible edit sequences and the user picks one that he considers reasonable without knowing whether this actually results in a consistent model or the desired model.

Learning, Incrementality and Invocation "*Learning* is the ability of a program to improve its performance on a given task over time" (Russell & Norvig 2010). It appears to be promising enhancement for improving the performance of search-based approaches (Barriga et al. 2018) and can help to identify hidden policies and user preferences (Barriga et al. 2020; Ludovico et al. 2020).

The feature *incrementality* refers to the ability of the repair function to make use of the (side) results of previous invocations. This has been a goal works in the model repair domain, (Reder & Egyed 2012; Giese & Wagner 2009; Mens et al. 2007; Cicchetti et al. 2012).

A repair may be invoked *on-demand* or *event-based*, e.g. upon every manual change, see also the classification restoration-based vs. propagation-based in (Anjorin et al. 2019).

6.8. Repair - Resolutions Feature

The other important abstract sub-feature of *Repair* are *Resolutions*, depicted in Fig. 14.

Results An important property is the *size* of the produced result set. A repair producing at most a *single* result is considered automatic. In the light of ambiguity, this means that the tools have to make decisions on its own, based on policies, metrics or randomness. *Multiple* results imply user interaction in the form of result selection, see Sect. 6.7. The tool can leverage this activity by *ordering* (Macedo et al. 2017) the results. This requires an underlying metric to compare results, e.g. induced by edits' costs, policies, or inconsistency levels. An interesting approach is taken in JTL (Cicchetti et al. 2011), which is able to work with multiple results simultaneously over time. The result may be *presented* either as a change, see Sect. 6.3, or in a simplified form by only presenting the final *state* of the model. Optionally, the result may be abstract in a sense that it contains placeholders (e.g. for attribute values of newly created elements). In the database jargon they are termed *labeled nulls* (Arenas et al. 2014) as indication for the user to take further actions.

Properties Arguably, the most important *property* of a repair approach is *correctness*. Its three stages, originally defined in (Macedo et al. 2017), were formally described in Sect. 4: *well-behaved* (the results do not add new inconsistencies), consistency *improving* (the results are less inconsistent than the input), and *fully* consistency restoring (the results are completely consistent). The repair is called *complete* (Hermann et al. 2011) when the repair function produces at least one result for every input. The principle of *hippocraticness* goes back to

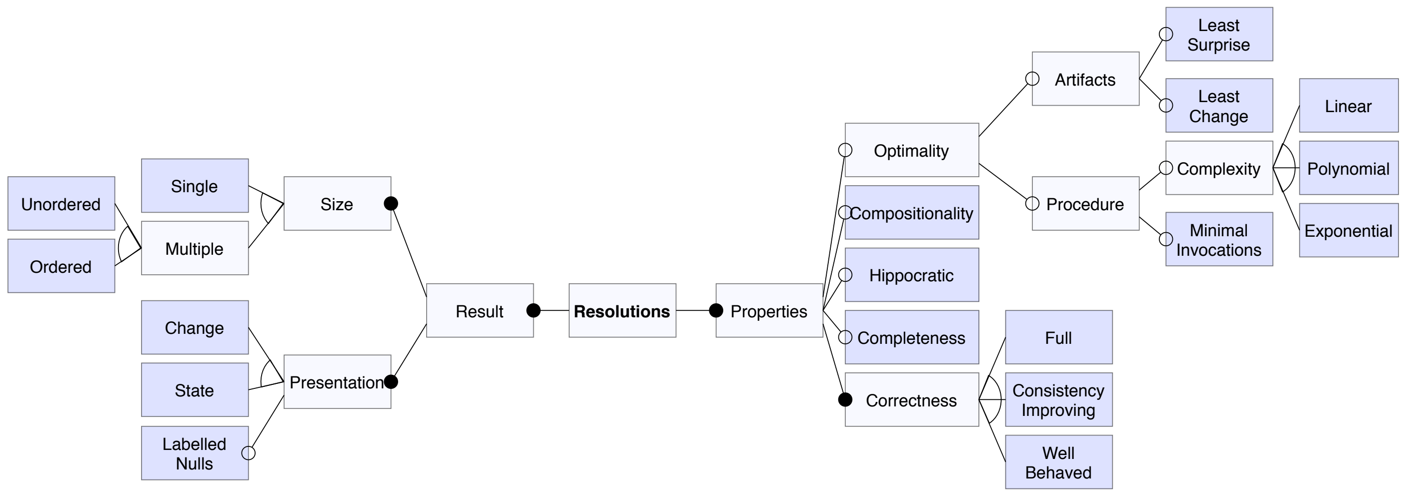


Figure 14 *Repair - Resolutions* Feature

Stevens (Stevens 2008) describing that when repair invoked on a consistent (multi-)model it produces the idle update ("does no harm"). *Compositionality*, which plays a central role in many theoretical considerations and the lens framework (Foster et al. 2007; Diskin et al. 2010; Johnson & Rosebrugh 2016), refers to the preservation of repair properties when they are composed into bigger repair functions. Finally, we distinguish between two aspects of *optimality*, i.e. concerning the produced *artifacts* on the one hand and the *procedure* itself on the other hand. In Sect. 4, we identified ambiguity as a core challenge in repair. Both fully automatic and semi-automatic tools thus require a metric to compare the quality of results beyond consistency. *Least Change* is a quantifiable metric proposed in (Macedo & Cunha 2016) based on the edit distance between two models. Since least change arguably not always coincide with the preferred choice, (Cheney et al. 2015) proposed the qualitative notion of *least surprise*. Optimality of the procedure is usually tantamount with efficiency. In computer science it is common to compare algorithms based on *complexity* w.r.t. runtime and memory consumption. Another notion of optimality was recently given in (Stevens 2020), which considers a composite repair procedure optimal when it requires a minimal invocation of elementary repair actions.

6.9. Feature Model Application: Existing Tools

Finally, we want to apply our feature model. For this we selected four model management tools (*eMoflon* (Leblebici et al. 2014), *Echo* (Macedo et al. 2014), *Epsilon* (EVL+Strace approach) (Samimi-Dehkordi et al. 2018), and *NMF* (Hinkel 2018)) and classified them using our feature model. The result is shown in Tab. 1, where each row contains the concrete characteristics for the respective abstract feature and tool.

We picked these four tools as an exemplary selection because they have recent publications and are actively maintained. Furthermore, they are based on rather different theoretical foundations and methodologies.

The tool eMoflon is the most recent culmination of tool development in the TGG research domain (Schürr 1994; Giese & Wagner 2009; Hermann et al. 2011). It is a chosen representative for the class of graph transformation based tools such as MoTe (Giese & Wagner 2009), Henshin (Biermann et al. 2008), etc.

Echo is model transformation and synchronization tool based OCL and QVT-r syntax built on top of the Alloy model finder. It was one of the representative examples of the study in (Macedo et al. 2017) and it is a representative for solver-based tools such as e.g. JTL (Cicchetti et al. 2011), *mediniQVT* (Anjorin et al. 2017), etc.

Epsilon (Paige et al. 2009) is a model management framework for EMF comprising a set of DSLs for various model management tasks. The foundation is an object-oriented language for model manipulation called *Epsilon Object Language (EOL)*. Consistency and repair facilities are provided by the *Epsilon Validation Language (EVL)* (D. Kolovos et al. 2008), which allows to define consistency rules declaratively in an OCL-like language and optionally augment them with quick fixes, i.e. procedural EOL programs. In (Samimi-Dehkordi et al. 2018), the authors developed a rigorous process for multi-model repair with EVL and domain specific traceability models. This tools/approach stands for the class of tools where repair is carried out with tight user interaction. Other approaches in this class are e.g. *Xlinkit* (Nentwich et al. 2003), *Beanbag* (Xiong et al. 2009).

NMF is an MDE library and internal DSL for .NET providing code generation facilities. It is based on the EMF file format and built on a concept of incremental computations. Further, it comprises a bidirectional synchronization framework influenced by the algebraic lens framework (Hinkel & Burger 2019). NMF is a representative for (functional) programming-based approaches based on mathematical frameworks, such as e.g. *BiGUL* (Ko et al. 2016), *Boomerang* (Foster et al. 2007), etc.

7. Findings

During the design of our feature model and review of the associated literature, we collected a list of issues in model management. A lot of these issues have been reported by other researchers before but still persist to the present day.

	eMoflon	Echo	Epsilon	NMF
Models				
TechSpace	EMF, JVM	EMF	EMF, Other (EOL)	.NET, EMF
Formalism	Graphs	Logical	OO	Categorical
Changes				
Representation	Structural-Delta	State-Based	State-Based	Operational-Delta
Definition	Fixed	Customizable	Customizable	Customizable
Types	Rename, Insert, Update, Delete, Move	Insert, Delete	Insert, Update, Delete, Move	Rename, Insert, Update, Delete, Move, Split, Merge
Recording	Offline, Change Identification	Offline	Offline	Online (non-intrusive)
Meta-Information	Previous State	n/a	n/a	Full History
Consistency				
Definition	Grammar	Builtin, OCL (QVTr)	Builtin, EVL (OCL-like)	Combinator Library
Invocation	on-demand	event-based	event-based	event-based
Reporting	boolean	elements	elements	elements
Inconsistency Level	n/a	n/a	fixed (2)	n/a
Meta-Information	repair hint	n/a	repair hint	repair hint, scope
Formats				
Conformance	Typing, Constraints	Constraints	Typing, Constraints	Typing
Representation	Type Graph	Knowledge Base	Metamodel	Type System
Migration	n/a	n/a	n/a	n/a
Multi-Models				
Corr. Representation	Traces	Traces	Traces	Transformations
Corr. Definition	Formulas	Formulas	Formulas	Formulas
Arity	binary	multi-ary	multi-ary	binary
Informational Overlap	symmetric	symmetric	symmetric	symmetric
Paradigms	heterogeneously typed	heterogeneously typed	heterogeneously typed	homogeneous
Synthesis	n/a	Merging	Language Extension	n/a
Authority	n/a	n/a	n/a	n/a
Privacy	n/a	n/a	n/a	n/a
Communication	n/a	n/a	n/a	n/a
Concurrency	serial (parallel?)	serial	serial	serial
Repair				
Implementation	Declarative Rules (Syntactical, Semantical, Generative), Optimization (for matching)	SMT Solver (alloy/kodkod)	Operational Rules	Universal Solutions, Operational Rules
Human Interaction	n/a	Cost Specification, Result Selection	Strategy Selection	n/a
Incrementality	persisted traces	n/a	persisted traces	incremental computation
Results	single	multiple (ordered)	single	single
Properties	correctness, completeness	correctness, hippocraticness, least change	none	correctness, hippicraticness, compositionality

Table 1 Tool classification

Standards A noteworthy share of tools, e.g. most of the ones mentioned in (Hidaka et al. 2016), is no longer retrievable because of broken URLs, non existing source code repositories, or dependencies towards outdated Eclipse versions, which are no longer runnable on recent operating systems. Thus, there is a need for standardized tool repositories and benchmarks for making evaluations reproducible. Promising endeavors in this direction have already started, see e.g. (Anjorin et al. 2019; Basciani et al. 2014), but have to be pursued further. Regarding file formats, EMF has become the de-facto standard in the MDE community for storing metamodels and models. However, yet there is no commonly agreed upon standard for expressing model modifications because changes are conceived differently (state-based vs. delta-based). Version control systems express changes as line insertions and removals, which is generally not well aligned with EMF/XMI semantics and necessitates change identification procedures. We argue that the delta-based change representation is superior over the state-based representation since deltas convey more information and allow strict traceability, which is generally lost in the state-based case. Thus, a technical standard for EMF file versioning and modification representation is paramount for model management and first attempts in this direction have already started (Yohannis et al. 2019). Regarding the definition of consistency rules, OCL (and its variations) are the most common means but their use is not equally as widespread as the EMF format. It is even more complicated when it comes to global consistency of multiple inter-related models.

Management of Multi-Models We analyzed the nature of multi-models and extended the feature model in (Macedo et al. 2017) by making multi-models a dimension of its own right. Multi-models are based on a correspondence relation, which can be represented in different ways: An underlying system model, cross-reference links, model transformations or model enhancement. Thus, there is also no standard for the definition and encoding of correspondences, which makes it challenging to compare update propagation tools, see (Anjorin et al. 2019). QVT-r (Object Management Group 2016a) has once been proposed as a possible standard but never got a widespread acceptance due to its semantic issues (Stevens 2008). Aside from this, other aspects such as authority, privacy, concurrency and issues with network communication are less investigated in the model management literature so far.

Integration of approaches We identified clusters of features often used in combination. All of the rule-based approaches show a tight connection between the consistency rule definition and repair implementation, which is established via repair hints. Those approaches, which utilize a more abstract logical formulation of consistency rules are more likely to implement model repair in a search-based way. The different dimensions (human interaction, learning, rule- and search-based implementations) of repair approaches show that they are not competing with each other but can instead be worthwhile to combine. Notable attempts can be found in the *ModelAnalyzer* (Reder & Egyed 2012) (combines human interaction and rule-based repair), *eMoflon* (Leblebici et al. 2014) (combines search- and rule-based approaches) or the approach in (Kessentini et al. 2018) (combines a search-based approach with human interaction).

The repair problem itself The general repair problem is NP-complete (complexity of general search problems) or even undecidable (think of contradicting consistency rules), i.e. having no solution. In general completely automated repair is not possible. Hence, it may be more worthwhile to focus further research in this direction on particular domains where one can harness domain dependent expert knowledge, which can help to find "the best" solution. See also the conclusions in the report on the least surprise principle (Cheney et al. 2015).

8. Conclusion, Related and Future Work

Our work is directly related to the major surveys in the fields of model repair, update propagation/bx, multi-viewpoint modeling and model migration. While those studies are of secondary nature, our work can be considered *tertiary* since it is based on a thorough study of the above. The added value in this paper is a refined feature model that combines the different aspects of model management and allows to compare more distinct approaches and tools. It is important to note that our model further extends the model in (Macedo et al. 2017) especially w.r.t. the implementation of model repair identifying universal solutions as a third approach besides rule- and search-based approaches.

For the near future, we plan to apply our feature model to many more tools and approaches mentioned in the literature. Furthermore, we plan to investigate the format feature and model migration tools in more detail, since this dimension has received a less thorough treatment in this paper. Also, we plan to use this feature model to develop a tool infrastructure for model management in the spirit of (Anjorin et al. 2019), which allows the integration of heterogeneous model repair tools. With this regard, the abstract megamodel build system (Stevens 2020) can be a promising candidate for the technological foundation.

Acknowledgments

We would like to thank all the anonymous reviewers for their helpful comments, which helped to improve the quality of this paper substantially.

References

Aizenbud-Reshef, N., Nolan, B. T., Rubin, J., & Shaham-Gafni, Y. (2006). Model traceability. *IBM Systems Journal*, *45*(3), 515–526. doi: 10.1147/sj.453.0515

Alanen, M., & Porres, I. (2003). Difference and union of models. In P. Stevens, J. Whittle, & G. Booch (Eds.), *Uml 2003 - the unified modeling language, modeling languages and applications, 6th international conference* (Vol. 2863, pp. 2–17). Springer. doi: 10.1007/978-3-540-45221-8_2

Ambler, S. W., & Sadalage, P. J. (2006). *Refactoring Databases: Evolutionary Database Design (paperback)*. Pearson Education.

Anjorin, A., Buchmann, T., Westfechtel, B., Diskin, Z., Ko, H.-S., Eramo, R., … Zündorf, A. (2019, September). Benchmarking bidirectional transformations: theory, implementation, application, and assessment. *Software and Systems Modeling*. doi: 10.1007/s10270-019-00752-x

Anjorin, A., Diskin, Z., Jouault, F., Ko, H.-S., Leblebici, E., & Westfechtel, B. (2017). Benchmarx reloaded: A practical benchmark framework for bidirectional transformations. In *BX@ETAPS 2017*.

Antkiewicz, M., & Czarnecki, K. (2008). Design Space of Heterogeneous Synchronization. In R. Lämmel, J. Visser, & J. Saraiva (Eds.), *GTTSE 2007* (pp. 3–46). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-540-88643-3_1

Arenas, M., Barceló, P., Libkin, L., & Murlak, F. (2014). *Foundations of Data Exchange*. Cambridge: Cambridge University Press. doi: 10.1017/CBO9781139060158

Atkinson, C., & Kühne, T. (2001). The Essence of Multilevel Metamodeling. In M. Gogolla & C. Kobryn (Eds.), *UML 2001* (pp. 19–33). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-45441-1_3

Atkinson, C., Stoll, D., & Bostan, P. (2010). Orthographic Software Modeling: A Practical Approach to View-Based Development. In L. A. Maciaszek, C. González-Pérez, & S. Jablonski (Eds.), *Enase 2009* (pp. 206–219). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-14819-4_15

Bancilhon, F., & Spyratos, N. (1981). Update Semantics of Relational Views. *ACM Trans. Database Syst.*, *6*(4), 557–575.

Barbosa, D. M., Cretin, J., Foster, N., Greenberg, M., & Pierce, B. C. (2010). Matching Lenses: Alignment and View Update. In *ICFP '10* (pp. 193–204). New York, NY, USA: ACM. doi: 10.1145/1863543.1863572

Barr, M., & Wells, C. (1990). *Category theory for computing science*. Prentice Hall.

Barriga, A., Rutle, A., & Heldal, R. (2018). Automatic model repair using reinforcement learning. In *Proceedings of MODELS 2018 workshops: Modcomp, mrt, ocl, flexmde, exe, commitmde, mdetools, gemoc, morse, mde4iot, mdebug, modevva, me, multi, hufamo, ammore, PAINS co-located with ACM/IEEE 21st international conference on model driven engineering languages and systems (MODELS 2018), copenhagen, denmark, october, 14, 2018.* (pp. 781–786). Retrieved from http://ceur-ws.org/Vol-2245/ammore_paper_1.pdf

Barriga, A., Rutle, A., & Heldal, R. (2020, July). Improving model repair through experience sharing. *Journal of Object Technology*, *19*(2), 13:1-21. (The 16th European Conference on Modelling Foundations and Applications (ECMFA 2020)) doi: 10.5381/jot.2020.19.2.a13

Basciani, F., Rocco, J. D., Ruscio, D. D., Salle, A. D., Iovino, L., & Pierantonio, A. (2014). MDEForge: an Extensible Web-Based Modeling Platform. In R. F. Paige, J. Cabot, M. Brambilla, L. M. Rose, & J. H. Hill (Eds.), *CloudMDE@MoDELS 2014* (Vol. 1242, pp. 66–75). CEUR-WS.org. Retrieved 2020-03-24, from http://ceur-ws.org/Vol-1242/paper10.pdf

Bennani, S., Ebersold, S., El Hamlaoui, M., Coulette, B., & Nassar, M. (2019, June). A Collaborative Decision Approach for Alignment of Heterogeneous Models. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)* (pp. 112–117). (ISSN: 2641-8169) doi: 10.1109/WETICE.2019.00032

Bergmann, G., Ráth, I., Varró, G., & Varró, D. (2012, July). Change-driven model transformations. *Software & Systems Modeling*, *11*(3), 431–461. doi: 10.1007/s10270-011-0197-9

Bernstein, P. A. (2003). Applying Model Management to Classical Meta Data Problems. In *CIDR*.

Bézivin, J., Jouault, F., Rosenthal, P., & Valduriez, P. (2005). Modeling in the Large and Modeling in the Small. In U. Aßmann, M. Aksit, & A. Rensink (Eds.), *Model driven architecture: European mda workshops: Foundations and applications, mdafa 2003 and mdafa 2004* (pp. 33–46). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/11538097_3

Biermann, E., Ermel, C., & Taentzer, G. (2008). Precise Semantics of EMF Model Transformations by Graph Transformation. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, & M. Völter (Eds.), *Model 2008* (pp. 53–67). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-87875-9_4

Boronat, A., Knapp, A., Meseguer, J., & Wirsing, M. (2009). What Is a Multi-modeling Language? In *Wadt 2008* (pp. 71–87). Berlin, Heidelberg: Springer Berlin Heidelberg.

Boronat, A., & Meseguer, J. (2010, May). An algebraic semantics for MOF. *Formal Aspects of Computing*, *22*(3), 269–296. Retrieved 2020-03-26, from https://doi.org/10.1007/s00165-009-0140-9 doi: 10.1007/s00165-009-0140-9

Brambilla, M., Cabot, J., & Wimmer, M. (2017). *Model-Driven Software Engineering in Practice* (2nd ed.). Morgan & Claypool Publishers.

Bruneliere, H., Burger, E., Cabot, J., & Wimmer, M. (2019, June). A feature-based survey of model view approaches. *Software & Systems Modeling*, *18*(3), 1931–1952. doi: 10.1007/s10270-017-0622-9

Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., & Sabetzadeh, M. (2006). A Manifesto for Model Merging. In *GaMMa '06* (pp. 5–12). New York, NY, USA: ACM. doi: 10.1145/1138304.1138307

Buchmann, T. (2019, October). BXtend - A Framework for (Bidirectional) Incremental Model Transformations. In *Modelsward 2019 proceedings* (pp. 336–345). doi: 10.5220/0006563503360345

Bézivin, J. (2004). In search of a Basic Principle for Model-Driven Engineering. *Novatica – Special Issue on UML (Unified Modeling Language)*, *5*(2), 21–24. Retrieved 2020-08-11, from https://hal.archives-ouvertes.fr/hal-00442702

Bézivin, J. (2005, May). On the unification power of models. *Software & Systems Modeling*, *4*(2), 171–188. Retrieved 2020-06-29, from https://doi.org/10.1007/s10270-005-0079-0 doi: 10.1007/s10270-005-0079-0

Bézivin, J., Jouault, F., & Valduriez, P. (2004, October). On the Need for Megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applica-*

tions,(2004). Vancouver, Canada. Retrieved 2019-09-05, from https://hal.archives-ouvertes.fr/hal-01222947

Cheney, J., Gibbons, J., McKinna, J., & Stevens, P. (2015). Towards a Principle of Least Surprise for Bidirectional Transformations. *Proceedings of the 4th International Workshop on Bidirectional Transformations co-located with Software Technologies: Applications and Foundations (STAF 2015)*, *1396*, 66–80.

Cicchetti, A., Ciccozzi, F., & Leveque, T. (2012). Supporting Incremental Synchronization in Hybrid Multi-view Modelling. In J. Kienzle (Ed.), *Models in Software Engineering* (pp. 89–103). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-29645-1_11

Cicchetti, A., Ciccozzi, F., & Pierantonio, A. (2019, December). Multi-view approaches for software and system modelling: a systematic literature review. *Software and Systems Modeling*, *18*(6), 3207–3233. doi: 10.1007/s10270-018-00713-w

Cicchetti, A., Di Ruscio, D., Eramo, R., & Pierantonio, A. (2011). JTL: A Bidirectional and Change Propagating Transformation Language. In B. Malloy, S. Staab, & M. van den Brand (Eds.), *Sle 2011* (pp. 183–202). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-19440-5_11

Cleve, A., Kindler, E., Stevens, P., & Zaytsev, V. (2019). Multi-directional Transformations and Synchronisations (Dagstuhl Seminar 18491). *Dagstuhl Reports*, *8*(12), 1–48. doi: 10.4230/DagRep.8.12.1

Codd, E. F. (1970, June). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, *13*(6), 377–387. doi: 10.1145/362384.362685

Cohn, P. M. (1965). *Universal Algebra* (1st edition ed.). Harper & Row.

Courcelle, B. (1997). The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg (Ed.), *Handbook of graph grammars and computing by graph transformation* (pp. 313–400). River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Czarnecki, K., Foster, N., Hu, Z., Lämmel, R., Schürr, A., & Terwilliger, J. F. (2009). Bidirectional Transformations: A Cross-Discipline Perspective. In *Icmt 2009* (pp. 193–204).

de Lara, J., Guerra, E., Kienzle, J., & Hattab, Y. (2018, October). Facet-oriented modelling: open objects for model-driven engineering. In *Sle 2018* (pp. 147–159). Boston, MA, USA: Association for Computing Machinery. doi: 10.1145/3276604.3276610

Di Ruscio, D., Eramo, R., & Pierantonio, A. (2012). Model Transformations. In M. Bernardo, V. Cortellessa, & A. Pierantonio (Eds.), *SFM 2012* (pp. 91–136). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-30982-3_4

Diskin, Z. (2009). Model Synchronization: Mappings, Tiles, and Categories. In J. M. Fernandes, R. Lämmel, J. Visser, & J. Saraiva (Eds.), *GTTSE 2009* (pp. 92–165). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-18023-1_3

Diskin, Z., Gholizadeh, H., Wider, A., & Czarnecki, K. (2016, January). A three-dimensional taxonomy for bidirectional model synchronization. *J. Syst. Softw*, *111*, 298–322. doi: 10.1016/j.jss.2015.06.003

Diskin, Z., Kokaly, S., & Maibaum, T. (2013). Mapping-aware megamodeling: Design patterns and laws. *LNCS*, *8225*, 322–343. doi: 10.1007/978-3-319-02654-1_18

Diskin, Z., König, H., & Lawford, M. (2018). Multiple Model Synchronization with Multiary Delta Lenses. In A. Russo & A. Schürr (Eds.), *FASE'18 proceedings* (pp. 21–37). Springer International Publishing.

Diskin, Z., König, H., & Lawford, M. (2019, November). Multiple model synchronization with multiary delta lenses with amendment andK-Putput. *Formal Aspects of Computing*, *31*(5), 611–640. doi: 10.1007/s00165-019-00493-0

Diskin, Z., Maibaum, T., & Czarnecki, K. (2012). Intermodeling, Queries, and Kleisli Categories. In J. de Lara & A. Zisman (Eds.), *Fundamental Approaches to Software Engineering* (pp. 163–177). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-28872-2_12

Diskin, Z., Maibaum, T., & Czarnecki, K. (2015). A Model Management Imperative: Being Graphical Is Not Sufficient, You Have to Be Categorical. In G. Taentzer & F. Bordeleau (Eds.), *Modelling Foundations and Applications* (pp. 154–170). Cham: Springer International Publishing. doi: 10.1007/978-3-319-21151-0_11

Diskin, Z., & Wolter, U. (2007). A diagrammatic logic for object-oriented visual modeling. In *Accat '07* (pp. 19–41).

Diskin, Z., Xiong, Y., & Czarnecki, K. (2010). From State-to Delta-Based Bidirectional Model Transformations. In L. Tratt & M. Gogolla (Eds.), *Theory and Practice of Model Transformations* (pp. 61–76). Springer Berlin Heidelberg.

Diskin, Z., Xiong, Y., & Czarnecki, K. (2011). Specifying Overlaps of Heterogeneous Models for Global Consistency Checking. In *Mdi@models 2010* (pp. 165–179).

Egyed, A. (2007). Fixing inconsistencies in UML design models. *Proceedings - International Conference on Software Engineering*, 292–301. doi: 10.1109/ICSE.2007.38

Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., & Taentzer, G. (2007). Information Preserving Bidirectional Model Transformations. In M. B. Dwyer & A. Lopes (Eds.), *Fundamental Approaches to Software Engineering* (pp. 72–86). Springer Berlin Heidelberg.

Ehrig, H., Ehrig, K., & Hermann, F. (2008, June). From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. *Electronic Communications of the EASST*, *10*(0). Retrieved 2018-08-30, from https://journal.ub.tu-berlin.de/eceasst/article/view/154 doi: 10.14279/tuj.eceasst.10.154

Ehrig, H., Ehrig, K., Prange, U., & Taentzer, G. (2006). *Fundamentals of algebraic graph transformation*. Springer.

El Hamlaoui, M., Bennani, S., Nassar, M., Ebersold, S., & Coulette, B. (2018, March). A MDE Approach for Heterogeneous Models Consistency. In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering* (pp. 180–191). Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, Lda. Retrieved 2020-06-29, from https://doi.org/10.5220/0006774101800191 doi: 10.5220/0006774101800191

Engels, G., Hausmann, J. H., Heckel, R., & Sauer, S. (2000). Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In A. Evans, S. Kent, & B. Selic (Eds.), *UML 2000 — The Unified Modeling Language* (pp. 323–337). Berlin, Heidelberg: Springer. doi: 10.1007/3-540-40011-7_23

Eramo, R., Malavolta, I., Muccini, H., Pelliccione, P., & Pierantonio, A. (2012, February). A model-driven approach to automate the propagation of changes among Architecture Description Languages. *Software & Systems Modeling*, *11*(1), 29–53. doi: 10.1007/s10270-010-0170-z

Eramo, R., Pierantonio, A., & Rosa, G. (2015, October). Managing uncertainty in bidirectional model transformations. In *SLE 2015* (pp. 49–58). Pittsburgh, PA, USA: Association for Computing Machinery. doi: 10.1145/2814251.2814259

Favre, J.-M., & NGuyen, T. (2005). Towards a Megamodel to Model Software Evolution Through Transformations. *Electronic Notes in Theoretical Computer Science*, *127*(3), 59–74. doi: https://doi.org/10.1016/j.entcs.2004.08.034

Feldmann, S., Kernschmidt, K., Wimmer, M., & Vogel-Heuser, B. (2019, July). Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering. *Journal of Systems and Software*, *153*, 105–134. doi: 10.1016/j.jss.2019.03.060

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., & Goedicke, M. (1992, March). Viewpoints: a framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, *02*(01), 31–57. (Publisher: World Scientific Publishing Co.) doi: 10.1142/S0218194092000038

Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., & Schmitt, A. (2007, may). Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-update Problem. *ACM Trans. Program. Lang. Syst.*, *29*(3). doi: 10.1145/1232420.1232424

Giese, H., & Wagner, R. (2009, February). From model transformation to incremental bidirectional model synchronization. *Software & Systems Modeling*, *8*(1), 21–43. Retrieved 2018-08-30, from https://doi.org/10.1007/s10270-008-0089-9 doi: 10.1007/s10270-008-0089-9

Gomes, C., Barroca, B., & Amaral, V. (2014, September). Classification of Model Transformation Tools: Pattern Matching Techniques. In *Model-Driven Engineering Languages and Systems* (pp. 619–635). Springer, Cham. Retrieved 2018-01-04, from https://link.springer.com/chapter/10.1007/978-3-319-11653-2_38 doi: 10.1007/978-3-319-11653-2_38

Gruschko, B., Kolovos, D., & Paige, R. (2007). Towards Synchronizing Models with Evolving Metamodels. In *Modse 2007*.

Habel, A., & Pennemann, K.-H. (2009, April). Correctness of high-level transformation systems relative to nested conditions†. *Mathematical Structures in Computer Science*, *19*(2), 245–296. doi: 10.1017/S0960129508007202

Hebig, R., Khelladi, D. E., & Bendraou, R. (2017, May). Approaches to Co-Evolution of Metamodels and Models: A Survey. *IEEE Transactions on Software Engineering*, *43*(5), 396–414. doi: 10.1109/TSE.2016.2610424

Hermann, F., Ehrig, H., Ermel, C., & Orejas, F. (2012). Concurrent Model Synchronization with Conflict Resolution Based on Triple Graph Grammars. In J. de Lara & A. Zisman (Eds.), *Fase 2012* (pp. 178–193). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-28872-2_13

Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., & Xiong, Y. (2011). Correctness of Model Synchronization Based on Triple Graph Grammar. In J. Whittle, T. Clark, & T. Kühne (Eds.), *Models 2011* (pp. 668–682). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-24485-8_49

Hidaka, S., Hu, Z., Inaba, K., Kato, H., & Nakano, K. (2011, November). GRoundTram: An integrated framework for developing well-behaved bidirectional model transformations. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)* (pp. 480–483). doi: 10.1109/ASE.2011.6100104

Hidaka, S., Tisi, M., Cabot, J., & Hu, Z. (2016, jul). Feature-based Classification of Bidirectional Transformation Approaches. *Softw. Syst. Model.*, *15*(3), 907–928.

Hinkel, G. (2018). NMF: A Multi-platform Modeling Framework. In A. Rensink & J. Sánchez Cuadrado (Eds.), *Theory and Practice of Model Transformation* (pp. 184–194). Cham: Springer International Publishing. doi: 10.1007/978-3-319-93317-7_10

Hinkel, G., & Burger, E. (2019, February). Change Propagation and Bidirectionality in Internal Transformation DSLs. *Softw. Syst. Model.*, *18*(1), 249–278. doi: 10.1007/s10270-017-0617-6

Hofmann, M., Pierce, B., & Wagner, D. (2012). Edit Lenses. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (pp. 495–508). New York, NY, USA: ACM. doi: 10.1145/2103656.2103715

Hohpe, G., & Woolf, B. (2012). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Pearson Education.

Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems* (2nd ed.). Cambridge: Cambridge University Press. doi: 10.1017/CBO9780511810275

ISO/IEC JTC 1/SC 7 Software and systems engineering. (2011, December). *Iso/iec/ieee 42010:2011 - systems and software engineering — architecture description.* https://www.iso.org/standard/50508.html.

Jackson, D. (2016). *Software Abstractions: Logic, Language, and Analysis*. The MIT Press.

Johnson, M., & Rosebrugh, R. (2007, December). Fibrations and universal view updatability. *Theoretical Computer Science*, *388*(1), 109–129. doi: 10.1016/j.tcs.2007.06.004

Johnson, M., & Rosebrugh, R. (2017). Symmetric delta lenses and spans of asymmetric delta lenses. *The Journal of Object Technology*, *16*(1), 2:1. doi: 10.5381/jot.2017.16.1.a2

Johnson, M., Rosebrugh, R., & Wood, R. (2010, March). Algebras and Update Strategies. *j-jucs*, *16*(5), 729–748. doi: 10.3217/jucs-016-05-0729

Johnson, M., & Rosebrugh, R. D. (2016). Unifying Set-Based, Delta-Based and Edit-Based Lenses. In A. Anjorin & J. Gibbons (Eds.), *Bx@ETAPS 2016* (Vol. 1571, pp. 1–13). CEUR-

WS.org.

Johnson, M., & Stevens, P. (2018, April). Confidentiality in the process of (model-driven) software development. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming* (pp. 1–8). Nice, France: Association for Computing Machinery. doi: 10.1145/3191697.3191714

Kehrer, T., Taentzer, G., Rindt, M., & Kelter, U. (2016). Automatically Deriving the Specification of Model Editing Operations from Meta-Models. In P. Van Gorp & G. Engels (Eds.), *Theory and Practice of Model Transformations* (pp. 173–188). Cham: Springer International Publishing. doi: 10.1007/978-3-319-42064-6_12

Kessentini, W., Sahraoui, H., & Wimmer, M. (2019, February). Automated metamodel/model co-evolution: A search-based approach. *Information and Software Technology*, *106*, 49–67. doi: 10.1016/j.infsof.2018.09.003

Kessentini, W., Wimmer, M., & Sahraoui, H. (2018, October). Integrating the Designer in-the-loop for Metamodel/Model Co-Evolution via Interactive Computational Search. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (pp. 101–111). New York, NY, USA: Association for Computing Machinery. Retrieved 2020-08-19, from https://doi.org/10.1145/3239372.3239375 doi: 10.1145/3239372.3239375

Kitchenham, B., Charters, S., Budgen, D., Brereton, P., Turner, M., Linkman, S., ... Visaggio, G. (2007, jul). *Guidelines for performing Systematic Literature Reviews in Software Engineering* (techreport No. 1). Software Engineering Group School of Computer Science and Mathematics Keele University Keele, Staffs ST5 5BG, UK: Evidence Based Software Engineering.

Klare, H., & Gleitze, J. (2019, September). Commonalities for Preserving Consistency of Multiple Models. In *MODELS 2019 companion* (pp. 371–378). doi: 10.1109/MODELS-C.2019.00058

Klare, H., Syma, T., Burger, E., & Reussner, R. (2019). A Categorization of Interoperability Issues in Networks of Transformations. *Journal of Object Technology*, *18*(3), 1–20. doi: 10.5381/jot.2019.18.3.a4.

Kleiner, M., Del Fabro, M. D., & Albert, P. (2010). Model Search: Formalizing and Automating Constraint Solving in MDE Platforms. In T. Kühne, B. Selic, M.-P. Gervais, & F. Terrier (Eds.), *Ecmfa 2010* (pp. 173–188). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-13595-8_15

Knapp, A., & Mossakowski, T. (2018). Multi-view Consistency in UML: A Survey. In *Graph Transformation, Specifications, and Nets* (pp. 37–60). Springer, Cham. doi: 10.1007/978-3-319-75396-6_3

Ko, H.-S., Zan, T., & Hu, Z. (2016). BiGUL: A Formally Verified Core Language for Putback-based Bidirectional Programming. In *PEPM '16* (pp. 61–72). ACM. doi: 10.1145/2847538.2847544

Kolovos, D., Paige, R., & Polack, F. (2008). Detecting and Repairing Inconsistencies Across Heterogeneous Models. In *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation* (pp. 356–364). Washington, DC, USA: IEEE Computer Society. Retrieved 2019-10-08, from https://doi.org/10.1109/ICST.2008.23 doi: 10.1109/ICST.2008.23

Kolovos, D. S., Di Ruscio, D., Pierantonio, A., & Paige, R. F. (2009). Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models* (pp. 1–6). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/CVSM.2009.5071714

König, H., & Diskin, Z. (2016). Advanced Local Checking of Global Consistency in Heterogeneous Multimodeling. In *Ecmfa 2016* (pp. 19–35). doi: 10.1007/978-3-319-42061-5_2

König, H., & Diskin, Z. (2017). Efficient Consistency Checking of Interrelated Models. In *Ecmfa 2017* (pp. 161–178). doi: 10.1007/978-3-319-61482-3_10

Kühne, T. (2006, December). Matters of (Meta-) Modeling. *Software & Systems Modeling*, *5*(4), 369–385. doi: 10.1007/s10270-006-0017-9

Lämmel, R. (2014). Coupled software transformations (Extended Abstract). In *Proceedings 1st international workshop on software evolution transformations* (pp. 31–35).

Leblebici, E., Anjorin, A., & Schürr, A. (2017). Inter-model Consistency Checking Using Triple Graph Grammars and Linear Optimization Techniques. In *Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering - Volume 10202* (pp. 191–207). New York, NY, USA: Springer-Verlag New York, Inc. doi: 10.1007/978-3-662-54494-5_11

Leblebici, E., Anjorin, A., & Schürr, A. (2014). Developing eMoflon with eMoflon. In D. Di Ruscio & D. Varró (Eds.), *Theory and Practice of Model Transformations* (pp. 138–145). Springer International Publishing. doi: 10.1007/978-3-319-08789-4_10

Ludovico, I., Barriga, A., Rutle, A., & Heldal, R. (2020, July). Model repair with quality-based reinforcement learning. *Journal of Object Technology*, *19*(2), 17:1-21. (The 16th European Conference on Modelling Foundations and Applications (ECMFA 2020)) doi: 10.5381/jot.2020.19.2.a17

Macedo, N., & Cunha, A. (2016, July). Least-change bidirectional model transformation with QVT-R and ATL. *Software & Systems Modeling*, *15*(3), 783–810. doi: 10.1007/s10270-014-0437-x

Macedo, N., Cunha, A., & Pacheco, H. (2014). Towards a framework for multidirectional model transformations. In *EDBT/ICDT 2014* (pp. 71–74).

Macedo, N., Jorge, T., & Cunha, A. (2017, July). A Feature-Based Classification of Model Repair Approaches. *IEEE Transactions on Software Engineering*, *43*(7), 615–640. doi: 10.1109/TSE.2016.2620145

Mantz, F., Taentzer, G., Lamo, Y., & Wolter, U. (2015). Co-evolving meta-models and their instance models: A formal approach based on graph transformation. *Science of Computer Programming*, *104*(1), 2–43. doi: 10.1016/j.scico.2015.01.002

Mens, T., Taentzer, G., & Runge, O. (2007, September).

Analysing refactoring dependencies using graph transformation. *Software & Systems Modeling*, *6*(3), 269–285. Retrieved 2020-06-29, from https://doi.org/10.1007/s10270-006-0044-6 doi: 10.1007/s10270-006-0044-6

Mens, T., & Van Der Straeten, R. (2007). Incremental Resolution of Model Inconsistencies. In J. L. Fiadeiro & P.-Y. Schobbens (Eds.), *Recent Trends in Algebraic Development Techniques* (pp. 111–126). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-71998-4_7

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. (2005, September). Challenges in software evolution. In *IWPSE'05* (pp. 13–22). doi: 10.1109/IWPSE.2005.7

Narayanan, A., Levendovszky, T., Balasubramanian, D., & Karsai, G. (2009). Automatic Domain Model Migration to Manage Metamodel Evolution. In A. Schürr & B. Selic (Eds.), *Model Driven Engineering Languages and Systems* (pp. 706–711). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-04425-0_57

Nassar, N., Kosiol, J., Arendt, T., & Taentzer, G. (2018). OCL2AC: Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules. In L. Lambers & J. Weber (Eds.), *Graph Transformation* (pp. 171–177). Cham: Springer International Publishing. doi: 10.1007/978-3-319-92991-0_11

Nassar, N., Radke, H., & Arendt, T. (2017). Rule-Based Repair of EMF Models: An Automated Interactive Approach. In E. Guerra & M. van den Brand (Eds.), *Theory and Practice of Model Transformation* (pp. 171–181). Springer International Publishing. doi: 10.1007/978-3-319-61473-1_12

Nentwich, C., Emmerich, W., & Finkelsteiin, A. (2003). Consistency Management with Repair Actions. In *ICSE '03* (pp. 455–464).

Nuseibeh, B., Easterbrook, S., & Russo, A. (2000, April). Leveraging inconsistency in software development. *Computer*, *33*(4), 24–29. doi: 10.1109/2.839317

Object Management Group. (2012). *Object Constraint Language (OCL) v.2.3.1.* Retrieved from http://www.omg.org/spec/OCL/2.3.1/

Object Management Group. (2014). *Business Process Model And Notation (BPMN) v.2.0.2.* Retrieved 14.01.2013, from http://www.omg.org/spec/BPMN

Object Management Group. (2015a). *Unified Modeling Language (UML) v.2.4.1.* Retrieved from http://www.omg.org/spec/UML

Object Management Group. (2015b). *XML Metadata Interchange (XMI) v.2.5.1.* Retrieved from https://www.omg.org/spec/XMI/2.5.1/

Object Management Group. (2016a). *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) v.1.3.* http://www.omg.org/spec/QVT/1.3. Retrieved from http://www.omg.org/spec/QVT/1.3

Object Management Group. (2016b). *Meta Object Facility (MOF) Core Specification v. 2.4.1.* Retrieved from http://www.omg.org/spec/MOF

Object Management Group. (2019). *Decision Model and Notation (DMN) v.1.2.* Retrieved 07.10.2019, from https://www.omg.org/spec/DMN/About-DMN/

Paige, R. F., Kolovos, D. S., Rose, L. M., Drivalos, N., & Polack, F. A. C. (2009). The Design of a Conceptual Framework and Technical Infrastructure for Model Management Language Engineering. In *Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems* (pp. 162–171). Washington, DC, USA: IEEE Computer Society. doi: 10.1109/ICECCS.2009.14

Paige, R. F., Matragkas, N., & Rose, L. M. (2016, January). Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. *Journal of Systems and Software*, *111*, 272–280. doi: 10.1016/j.jss.2015.08.047

Pearl, J. (1981). Heuristic Search Theory: Survey of Recent Results. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1* (pp. 554–562). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Pinna Puissant, J., Van Der Straeten, R., & Mens, T. (2012). Badger: A Regression Planner to Resolve Design Model Inconsistencies. In A. Vallecillo, J.-P. Tolvanen, E. Kindler, H. Störrle, & D. Kolovos (Eds.), *Modelling Foundations and Applications* (pp. 146–161). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-642-31491-9_13

Pinna Puissant, J., Van Der Straeten, R., & Mens, T. (2015, February). Resolving model inconsistencies using automated regression planning. *Software & Systems Modeling*, *14*(1), 461–481. Retrieved 2019-11-08, from https://doi.org/10.1007/s10270-013-0317-9 doi: 10.1007/s10270-013-0317-9

Pottinger, R. A., & Bernstein, P. A. (2003, January). Merging Models Based on Given Correspondences. In J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, & A. Heuer (Eds.), *Proceedings 2003 VLDB Conference* (pp. 862–873). San Francisco: Morgan Kaufmann. doi: 10.1016/B978-012722442-8/50081-1

Rabbi, F., Lamo, Y., & MacCaull, W. (2014, jan). Co-ordination of Multiple Metamodels, with Application to Healthcare Systems. *Procedia Computer Science*, *37*, 473–480. doi: 10.1016/J.PROCS.2014.08.071

Reder, A., & Egyed, A. (2010, September). Model/analyzer: a tool for detecting, visualizing and fixing design errors in UML. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (pp. 347–348). Antwerp, Belgium: Association for Computing Machinery. doi: 10.1145/1858996.1859069

Reder, A., & Egyed, A. (2012, September). Computing repair trees for resolving inconsistencies in design models. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (pp. 220–229). doi: 10.1145/2351676.2351707

Rivera, J. E., & Vallecillo, A. (2008). Representing and Operating with Model Differences. In R. F. Paige & B. Meyer (Eds.), *Objects, Components, Models and Patterns* (pp. 141–160). Springer Berlin Heidelberg. doi: 10.1007/978-3-540-69824-1_9

Roddick, J. F. (1992, December). Schema Evolution in Database Systems: An Annotated Bibliography. *SIGMOD Rec.*, *21*(4), 35–40.

Romero, J. R., Jaén, J. I., & Vallecillo, A. (2009). Realizing Correspondences in Multi-viewpoint Specifications. In *EDOC'09* (pp. 138–147). Piscataway, NJ, USA: IEEE Press. doi: 10.1109/EDOC.2009.23

Rose, L. M., Herrmannsdoerfer, M., Mazanek, S., Gorp, P. V., Buchwald, S., Horn, T., … Wimmer, M. (2014, February). Graph and model transformation tools for model migration. *Software & Systems Modeling*, *13*(1), 323–359. doi: 10.1007/s10270-012-0245-0

Rose, L. M., Kolovos, D. S., Paige, R. F., Polack, F. A. C., & Poulding, S. (2014, May). Epsilon Flock: a model migration language. *Software & Systems Modeling*, *13*(2), 735–755. Retrieved 2020-02-14, from https://doi.org/10.1007/s10270-012-0296-2 doi: 10.1007/s10270-012-0296-2

Rubin, J., & Chechik, M. (2013). N-way Model Merging. In *ESEC/FSE 2013* (pp. 301–311). New York, NY, USA: ACM. doi: 10.1145/2491411.2491446

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Rutle, A., Rossini, A., Lamo, Y., & Wolter, U. (2009). A Diagrammatic Formalisation of MOF-Based Modelling Languages. In *Tools europe 2009* (pp. 37–56). Springer, Berlin, Heidelberg.

Rutle, A., Rossini, A., Lamo, Y., & Wolter, U. (2012). A formal approach to the specification and transformation of constraints in MDE. *JLAMP*, *81*(4), 422–457.

Sabetzadeh, M., & Easterbrook, S. (2005). An Algebraic Framework for Merging Incomplete and Inconsistent Views. In *Re 2005* (pp. 306–315).

Saito, Y., & Shapiro, M. (2005, March). Optimistic replication. *ACM Computing Surveys*, *37*(1), 42–81. Retrieved 2020-03-05, from https://doi.org/10.1145/1057977.1057980 doi: 10.1145/1057977.1057980

Salay, R., Mylopoulos, J., & Easterbrook, S. (2009). Using Macromodels to Manage Collections of Related Models. In P. van Eck, J. Gordijn, & R. Wieringa (Eds.), *Advanced Information Systems Engineering* (pp. 141–155). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-02144-2_15

Samimi-Dehkordi, L., Zamani, B., & Kolahdouz-Rahimi, S. (2018, August). EVL+Strace: a novel bidirectional model transformation approach. *Information and Software Technology*, *100*, 47–72. doi: 10.1016/j.infsof.2018.03.011

Schürr, A. (1994). Specification of Graph Translators with Triple Graph Grammars. In *Wg '94* (pp. 151–163).

Segen, J. C. (1992). *The Dictionary of Modern Medicine*. CRC Press.

Silva, M. A. A. d., Mougenot, A., Blanc, X., & Bendraou, R. (2010, June). Towards Automated Inconsistency Handling in Design Models. In *Advanced Information Systems Engineering* (pp. 348–362). Springer, Berlin, Heidelberg. doi: 10.1007/978-3-642-13094-6_28

Spanoudakis, G., & Zisman, A. (2001). Inconsistency Management in Software Engineering: Survey and Open Research Issues. In *Handbook of software engineering and knowledge engineering* (pp. 329–380). doi: 10.1142/9789812389718_0015

Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*. Pearson Education.

Stevens, P. (2008, dec). Bidirectional model transformations in QVT: semantic issues and open questions. *Software & Systems Modeling*, *9*(1), 7. Retrieved from https://doi.org/10.1007/s10270-008-0109-9 doi: 10.1007/s10270-008-0109-9

Stevens, P. (2017, June). Bidirectional Transformations In The Large. In *Models 2017* (pp. 1–11). doi: 10.1109/MODELS.2017.8

Stevens, P. (2018). Towards Sound, Optimal, and Flexible Building from Megamodels. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (pp. 301–311). New York, NY, USA: ACM. doi: 10.1145/3239372.3239378

Stevens, P. (2020, March). Connecting software build with maintaining consistency between models: towards sound, optimal, and flexible building from megamodels. *Software and Systems Modeling*. doi: 10.1007/s10270-020-00788-4

Straeten, R. V. D., Puissant, J. P., & Mens, T. (2011). Assessing the Kodkod Model Finder for Resolving Model Inconsistencies. In *ECMFA*. doi: 10.1007/978-3-642-21470-7_6

Stünkel, P., König, H., Lamo, Y., & Rutle, A. (2018). Multimodel correspondence through inter-model constraints. In *Conference companion of the 2nd international conference on art, science, and engineering of programming* (p. 9–17). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3191697.3191715 doi: 10.1145/3191697.3191715

Stünkel, P., König, H., Lamo, Y., & Rutle, A. (2020). Towards multiple model synchronization with comprehensive systems. In *FASE 2020* (Vol. 12076). Springer, Cham. doi: 10.1007/978-3-030-45234-6_17

Tanenbaum, A. S., & Steen, M. v. (2007). *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall.

Vermolen, S., & Visser, E. (2008). Heterogeneous Coupled Evolution of Software Languages. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, & M. Völter (Eds.), *Model Driven Engineering Languages and Systems* (pp. 630–644). Berlin, Heidelberg: Springer. doi: 10.1007/978-3-540-87875-9_44

Xiong, Y., Hu, Z., Zhao, H., Song, H., Takeichi, M., & Mei, H. (2009). Supporting Automatic Model Inconsistency Fixing. In *ESEC/FSE '09* (pp. 315–324). New York, NY, USA: ACM. doi: 10.1145/1595696.1595757

Xiong, Y., Song, H., Hu, Z., & Takeichi, M. (2013, February). Synchronizing concurrent model updates based on bidirectional transformation. *Software & Systems Modeling*, *12*(1), 89–104. doi: 10.1007/s10270-010-0187-3

Yohannis, A., Rodriguez, R. H., Polack, F., & Kolovos, D. (2019, July). Towards Efficient Comparison of Change-Based Models. *Journal of Object Technology*, *18*(2), 7:1–21. doi: 10.5381/jot.2019.18.2.a7

About the authors

Patrick Stünkel is a Ph.D. research fellow at the Department of Computer science, Electrical engineering and Mathematical sciences at the Western Norway University of Applied Sciences (HVL) in Bergen, Norway. His research interests are Software (Multi) Modeling, Interoperability, System Integration

and Applications of Category Theory in Software Engineering. In his thesis he investigates means for expression and restoration of consistency in multi models. You can contact him at past@hvl.no.

Harald König is a professor for Computer Science at the University of Applied Sciences, FHDW Hannover, Germany, and Adjunct Professor at the Department of Computer science, Electrical engineering and Mathematical sciences at the Western Norway University of Applied Sciences (HVL), Bergen, Norway. Before entering academy, he worked at SAP in Walldorf and received his PhD in pure Mathematics from Leibniz University in Hannover, Germany. You can contact him at Harald.Koenig@fhdw.de.

Yngve Lamo holds a PhD in Computer Science, from the University of Bergen, Norway. Lamo is a professor at the Department of Computer science, Electrical engineering and Mathematical sciences at the Western Norway University of Applied Sciences (HVL), Bergen. His research interests span from formal foundations of Model Driven Software Engineering over applications to Health Informatics. He is leading the work package concerning the core clinical process in the Norwegian National cross-disciplinary research project INTROMAT (INtroducing personalized TReatment Of Mental health problems using Adaptive Technology). You can contact him at yla@hvl.no.

Adrian Rutle holds a PhD in Computer Science from the University of Bergen, Norway. Rutle is a professor at the Department of Computer science, Electrical engineering and Mathematical sciences at the Western Norway University of Applied Sciences (HVL), Bergen. Rutle's main interest is applying theoretical results from the field of model-driven software engineering to practical domains and has expertise in the development of modelling frameworks and domain-specific modelling languages. He also conducts research in the fields of modelling and simulation for robotics, eHealth, digital fabrication, smart systems and machine learning. You can contact him at aru@hvl.no.