A Complete Operator Library for DSL Evolution Specification

J.G.M. Mengerink[1], A. Serebrenik[1], R.R.H. Schiffelers[1,2], M.G.J. van den Brand[1]
{j.g.m.mengerink, a.serebrenik, r.r.h.schiffelers, m.g.j.v.d.brand}@tue.nl
ramon.schiffelers@asml.com
[1] Eindhoven University of Technology, The Netherlands,
[2] ASML, The Netherlands

***Abstract*— Domain-specific languages (DSLs) allow users to model systems using concepts from a specific domain. Evolution of DSLs triggers co-evolution of models developed in these languages. Manual co-evolution of the thousands of models is unfeasible, calling for an automated support.**

A prerequisite to automating model co-evolution with respect to DSL evolution is the ability to formally specify DSL evolution, *e.g.,* using predefined evolution operators. Success or failure of the practical application of the operator-based approach therefore depends heavily on the operators offered by the operator library at hand.

In this paper we evaluate the completeness of the state-of-the-art operator library claimed to be "practically complete" (which we denote as $\mathcal{H}$) by using it to specify evolution of an ecosystem of 22 commercial DSLs over the period of four years. We observe that 11% of the changes cannot be specified.

However, there is no guarantee that extending the library with the identified deficiencies will be sufficient to specify evolution of other DSLs. To mitigate this, we design a theoretically complete library of operators, $\mathcal{R}$. We observe that 77% of the operators from $\mathcal{R}$ are absent from $\mathcal{H}$. Of the deficiencies in $\mathcal{H}$, 72% could not be revealed by means of studying the extensive industrial ecosystem above.

Our study suggests that the existing operator libraries are not extensive enough to specify evolution of large model-driven software ecosystems. Since extending operator libraries on a per-case study basis does not yield satisfactory results so far, we advocate an alternative, *i.e.,*a theoretically complete library of operators $\mathcal{R}$.

I. Introduction

Domain specific (modeling) languages (DSLs) offer a way to model complex systems in terms of familiar domain concepts. DSLs are created by specifying their abstract syntax using meta-models [1] that define the concepts and structure of a language[1]. Because of this centralized definition of concepts and structure, meta-models have become, by design, a hotspot in the model driven software engineering (MDSE) development process.

As with classical software, meta-models change over time [2]. This means that when meta-models change, artifacts that depend on the meta-model must co-evolve to reflect changes made to the meta-models. This is known as the *co-evolution* problem in MDSE [3]. Artifacts that may require co-evolution include models [4], [5], [6], [7], model-transformations [8], [9], text editors, and graphical editors [10]. In industry, we observe that the number of artifacts conforming to a single meta-model number in the thousands, making manual co-evolution of artifacts time-consuming, error-prone, and therefore very costly. In practice, this results in a strongly reduced evolvability of (meta-)models and MDSE ecosystems in general.

To mitigate these challenges, we wish to automate the co-evolution of artifacts with respect to meta-model evolution. As already recognized by Meyers and Vangheluwe [11], **a first step towards this automation is the formalization of meta-model evolution. This formal evolution specification will then be the foundation on which future work for co-evolution will be grounded.**

An existing approach for formalizing meta-model evolution is the *operator-based*[2] approach [7], [13], in which the evolution of a meta-model is specified in terms of reusable operators that each encode an (often-occurring) pattern of evolution (*e.g.,* "Create new class"). The practical applicability of such an approach relies heavily on the set of available operators, known as *operator libraries*, such as those surveyed by Herrmannsdörfer *et al.* [14]. Our first research question is thus:

> **RQ1:** To what extent can existing operator libraries specify evolution of DSLs in a large-scale industrial MDSE ecosystem?

To answer **RQ1** we perform a conceptual replication of the work of Herrmannsdörfer *et al.*[14] and apply the state-of-the-art operator library to a large-scale industrial case study. We find that 11% of the evolutionary changes in the case study DSLs cannot be specified using the operator library, indicating a deficiency in the library. To mitigate this deficiency, we wish to extend the library. However, we aim beyond the scope of our case-study and wish to support any DSL based on `Ecore` [15], the main vehicle for the DSL design in the Eclipse Modeling Framework [16].

To achieve this goal two approaches can be considered: either to perform a series of case studies and incrementally extend the library to address the specific deficiencies identified in each case study, or to design an approach allowing one to specify evolution of DSLs in *any* case study. We advocate

[1]In our case study, there is a one-to-one correspondence between DSLs and meta-models, hence these terms are used as synonyms throughout this paper.

[2]Also known as operation-based approach [12].

the second approach. As opposed to *practical completeness* of the earlier work [14] the approach we propose results in a *theoretically complete* library, denoted $\mathcal{R}$ (Table V).

Based on $\mathcal{R}$, we study which operators from the complete library $\mathcal{R}$ are not supported in practice by means of our second research question.

> **RQ2:** Which operators from $\mathcal{R}$ are missing from state-of-the-art operator libraries?

To answer RQ1, we chose to use a top-down approach rather than a bottom-up approach. Aiming for a complete operator library, we wonder which operators we would not have uncovered using a bottom-up approach.

> **RQ3:** What are the limitations of our case study with respect to finding operator library deficiencies?

The remainder of this paper is structured as follows: In Section II we discuss related work. In Section III we present the setup of our study, and report on our results in Section IV. Lastly, in Section V we discuss our results, and we present limitations, and future work and conclusions in Sections VI and VII respectively.

II. Background

In the literature, several approaches have been presented towards automating the evolution of meta-models and co-evolution of meta-modeling artifacts (*e.g.,*models and model-transformation). In Section II-A, we give a brief introduction into the fundamental concepts of the `Ecore` meta-meta-model and meta-modeling itself. In Section II-B we discuss existing ways to specify evolution, and a number of different approaches to obtaining such evolution specifications are described in Section II-C. Lastly, we discuss existing operator libraries in Section II-D.

A. Meta-modeling

In model driven engineering (MDE), models abstract or simplify some part of a real life system [17], allowing analysis of that system. Meta-models describe the concepts and structures of these models, such that models are instances of meta-models just as objects are instances of classes. Consequently, models *conform* to their meta-models [18]. These relations are illustrated in Fig. 1 [19][17].

The structure of meta-models, in turn, is described by meta-meta-models, such as the `Ecore` meta-meta-model [15], which is part of the Eclipse Modeling Framework (EMF) [16], [20]. A fragment of the `Ecore` meta-meta-model is depicted in Fig. 2.

As with the relation of models with respect to meta-models, `Ecore`-based meta-models are instantiations of the `Ecore` meta-meta-model. In Fig. 3, an example of an `Ecore`-based meta-model is presented. This meta-model describes PetriNet models and consists of four instances

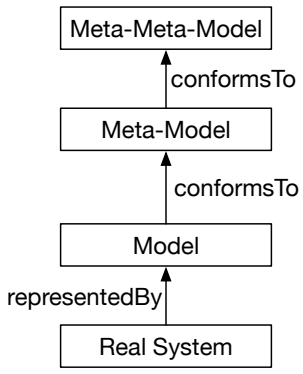


Fig. 1: Conformance relations between models, meta-models, and meta-meta-models.

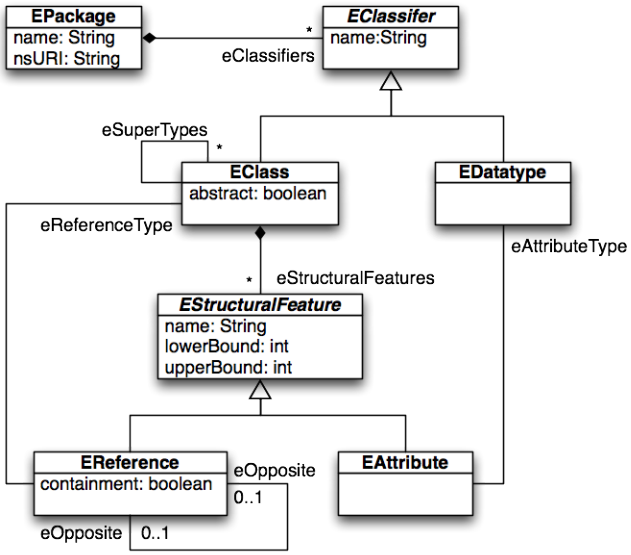


Fig. 2: A fragment of the Ecore meta-meta-model taken from the Eclipse Juno documentation [21].

of `EClasses`: PetriNet, Place, Arc, and Transition. They contain information using attributes (*e.g.,* Place.name), and are related using references. We distinguish two kinds of references: regular references (*e.g.,* Arc.place) and containment references (*e.g.,* PetriNet.transitions). Containment references are the `Ecore` equivalent of a composition association. Attributes and references are `EStructuralFeatures`.

B. How to Specify Meta-Model Evolution

The goal of *evolution specification* is to formalize the differences between two versions of a (meta-)model. In the literature, a number of approaches for the specification of (meta-)model evolution (or model differences in general) have been presented [22]:

Difference meta-models [22] define a language for specifying differences as models, according to the "everything is a model" principle. A model of this difference meta-model can represent the changes made to an initial version of a (meta-)model, in order to obtain the new version.

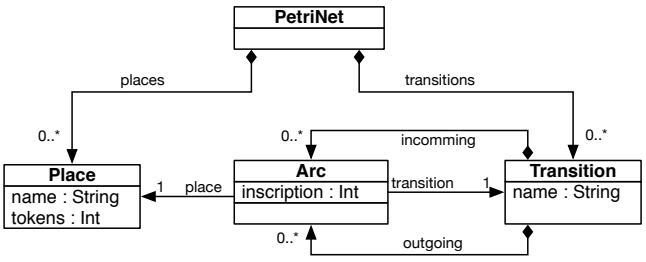
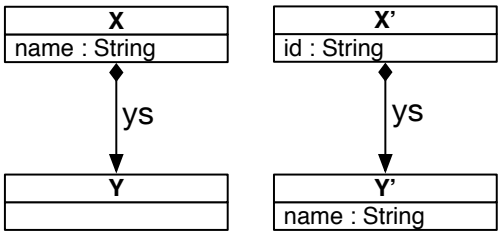


Fig. 3: An Ecore-based meta-model describing PetriNet models.



(a) Before Evolution (b) After Evolution

Fig. 4: An example of meta-model evolution

Coloring [23] techniques use a visual representation in which common parts of two (meta-)model versions are colored. However, this approach becomes less appealing as the (meta-)models under study become large.

Lastly, *Edit scripts* (also known as the directed delta [24], or operation-based approach [12]) formalize differences in terms of operations (such as `ADD` and `DELETE`) in a procedural way. An example of such an approach is the operator-based approach by Wachsmuth [7], which formalizes the differences between two meta-models in terms of reusable and configurable patterns of evolution known as *operators*.

C. How to Obtain Evolution Specification

In the literature, several approaches exist for obtaining such an evolution specification.

Manual Specification approaches [25] require the user to manually create the **model co-evolution** specification in some DSL. However, as meta-models and their evolutions become large, so does the effort and difficulty required to manually create co-evolution specification by hand [26]. For this reason, we consider *manual specification* to be out of scope of this research.

Recording approaches [13] record the editing performed to the meta-model by the user, and create an evolution specification based on these actions.

State-based differencing approaches [6], [13] take both the original, and the evolved version of a meta-model, and derive an evolution specification using model differencing.

By-example approaches [27] require the user to manually migrate several models between the old and new meta-model versions, and attempt to distill the meta-model changes.

To reduce the workload of the user, the mentioned approaches attempt to approximate an *evolution specification*, rather than have the user specify one by hand. However, this comes at a price: the approaches have to make certain choices for the user without confirming that those choices correspond to the user's intention. Consider Fig. 4. Different evolutionary scenarios could have taken place, including but not limited to (1) X.name was renamed to X'.id and Y'.name is new; (2) X.name was renamed to Y'.name and X'.id is new; (3) X.name was removed, and both X'.id and Y'.name are new. If an evolution specification is approximated, the algorithm that does so must choose one of these options, without knowing what the users intention was. We call this the *semantic gap*.

Lastly, *Operator-based* approaches [7], [13], in contrast to the other approaches, allow the user to specify the **meta-model evolution** by hand[3]. Rather than approximating the specification from a source of meta-data (*e.g.,* user actions on the meta-model), the user is asked to specify meta-model evolution in terms of reusable operators that each encode an often-occurring pattern of evolution (and possibly model co-evolution[4]). This mitigates the *semantic gap* that arises through *evolution specification approximation*, by having the user formally define the exact evolution. The operator-based approach has even been mathematically formalized in terms of graph transformations by Mantz *et al.* [28], [29].

D. Operator Libraries

Because the operator based approach allows for reuse of evolution patterns, it is appealing to large-scale industrial cases. However, the practical applicability of this approach relies heavily on the set of available operators [13].

A number of operator libraries have been proposed and applied to industrial case studies. Herrmannsdörfer *et al.* have taken a number of these libraries and case studies and combined them into a single library that provides 61 atomic and compound operators [14]. For each of these operators, a practical evaluation on eighteen case studies is provided. The authors claim that their library of 61 operators is complete for practical applications. Although the applicability of this operator library has already been investigated in an industrial context [14], these studies limit themselves to studies of a single meta-model at a time and are primarily set in the automotive domain. As we have explained at the start of Section III, our industrial case study of twenty-two DSLs with four years of history provides a more elaborate validation.

In this paper, we focus solely on *atomic* operators (also knows as primitives) [30], *i.e.,* operators with effects that cannot be decomposed into smaller operators on the meta-model. For example, the *compound* operator `CreateOppositeReference` [14] can be decomposed

[3]Some techniques even provide an initial version of the specification automatically

[4]In this paper we only consider evolution

into an application of `CreateReference` and an application of `SetOppositeReference`, where the two latter operators cannot be decomposed further. We have chosen to focus on atomic operators because they form the foundation for further operator specification (*i.e.,* every compound operator can be expressed in terms of atomic operators). In order to properly research compound operators (which we have marked as future work), a complete set of atomic operators is required first.

III. Study Setup

To answer **RQ1**, we perform a *conceptual replication* (*cf.* [31]) of the work of Herrmannsdörfer *et al.* [14], [32]. We apply the methodology of Herrmannsdörfer *et al.* [32] to investigate the extent to which the library described by Herrmannsdörfer *et al.* [14], which we will denote as $\mathcal{H}$, is able to specify the evolution of a large-scale industrial case study. Studies of to which extent library of operators covers a case-study have been previously performed in classical software engineering [33]. We opt for $\mathcal{H}$ as it is a state-of-the-art operator library that has been claimed to be "practically complete" [14].

While the earlier evaluations of operator libraries have been restricted to individual DSLs, we opt for an industrial DSL *ecosystem* (*cf.* [34], [35]). The ecosystem, known as Control Architecture Reference Model (CARM) [36], counts twenty-two interdependent DSLs with the corresponding models and model transformations [37], has a revision history of four years and is developed at ASML [38], provider of lithography equipment for the semiconductor industry.

To answer **RQ2**, we need to go beyond the specifics of an individual DSL. Therefore, we cannot use the approach taken by Herrmannsdörfer *et al.*, as its incremental nature strongly relies on case studies as the basis of the deficiency identification. Rather, we use the meta-meta-model as input to generate a complete library of atomic operators. We refer to this library as $\mathcal{R}$. Subsequently, we investigate to what extent the operators from $\mathcal{H}$ are present in $\mathcal{R}$.

To answer **RQ3**, we compare the answers to **RQ1** and **RQ2**, and determine which operator-library deficiencies were not discovered by our industrial case study.

Summarizing, to answer our research questions three comparisons are performed:

1) $\mathcal{H}$ with respect to our industrial case study (for **RQ1**);
2) $\mathcal{H}$ with respect to $\mathcal{R}$ (for **RQ2**);
3) the industrial case-study with respect to $\mathcal{R}$ (for **RQ3**).

A. Simplifying Analysis

Since $\mathcal{R}$ is complete, we can relate both $\mathcal{H}$ and the case study to $\mathcal{R}$, as illustrated in Fig. 5. Then, the answers to our research questions can be distilled as follows:

1) We identify EMFCompare differences (*e.g.,* d_5) related to an operator in $\mathcal{R}$ (*e.g.,* r_5) that is not related to any operator in $\mathcal{H}$. This indicates that specific EMFCompare difference (d_5) cannot be specified by $\mathcal{H}$. Identifying all such EMFCompare differences then answers **RQ1**.

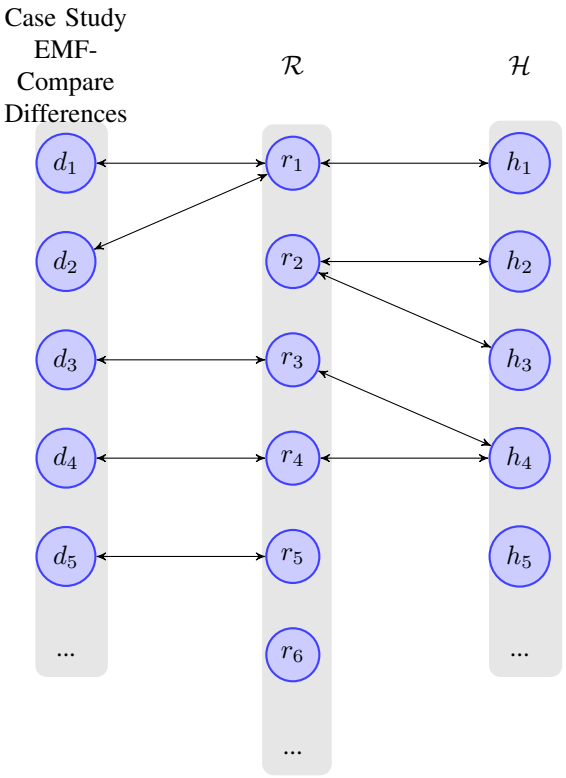


Fig. 5: Relations between EMFCompare-based case study differences, our complete library of atomic operators ($\mathcal{R}$), and the library of Herrmannsdörfer *et al.* [14] ($\mathcal{H}$).

2) To answer **RQ2**, we identify operators in $\mathcal{R}$ that are not related to any operator in $\mathcal{H}$ (*e.g.,* r_5, r_6 in Fig. 5). If this is the case, then $\mathcal{H}$ has a deficiency with respect to $\mathcal{R}$. Identifying all such deficiencies then answers **RQ2**.
3) To answer **RQ3**, we determine what atomic operators in $\mathcal{R}$ are not supported by $\mathcal{H}$, but are not revealed by the case study. To do so, we search for operators in $\mathcal{R}$, that are neither related to an EMFCompare difference, nor to an operator in $\mathcal{H}$, (*e.g.,* r_6 in Fig. 5). Such a pattern corresponds to a deficiency in the library that is not revealed by the case study.

In the remainder of this section, we discuss computation of our complete library of atomic operators $\mathcal{R}$ (Section III-B), extraction of the *evolution history* (Section III-C) and relating $\mathcal{H}$ to $\mathcal{R}$ (Section III-D).

B. Computing $\mathcal{R}$

In order to determine operator deficiencies that are not identified by a case study, we compute $\mathcal{R}$, a complete library of atomic operators. The primary requirement on a complete set of atomic operators is that it can specify every change possible on the meta-model. To this extent we need to be able to formalize meta-model changes. The EMFCompare difference model, for example, is able to encode every atomic meta-model change on `Ecore`-based meta-models in a difference model. As our case study is entirely `Ecore`-based, we construct our solution using the EMFCompare

difference model [39].

An atomic operator, by definition, should cause precisely one meta-model difference. Hence, we start by discussing meta-model differences identified by EMFCompare.

The EMFCompare difference model encodes each atomic change as so-called *Diff-Objects*. For the sake of simplicity, we view these Diff-Objects as four-tuples $\langle \delta, \phi, \epsilon, \nu \rangle$ where

- δ is a difference kind: `ADD`, `DELETE`, or `CHANGE` [5];
- ϕ is a meta-model element on which the change has been performed;
- ϵ is a structural feature of ϕ (*i.e.,* an attribute or reference, here and elsewhere we refer to elements of the `Ecore` meta-model fragment shown on Fig. 2);
- ν is a value to be added/deleted/changed (the type ν^* of ν should correspond to the datatype of ϵ).

Example III.1. *Changing the name of the class "Vertex" to "Node" can be recorded as* $\langle$`CHANGE`, *Vertex*, *name*, *"Node"*$\rangle$. *"Node" is of type String corresponding to the datatype of name (*cf. *Fig. 2).* □

Since EMFCompare can describe any change, intuitively we can generate all possible operators by iterating over every possible combination of the difference kind δ, type of ϕ (which corresponds to every element in the meta-meta-model), feature ϵ of that ϕ, and datatype ν^* of ϵ.

Example III.2. *The operator corresponding to the change in Example III.1 is* $\langle$`CHANGE`, `EClass`, *name*, `String`$\rangle$. *We also say that the change in Example III.1 is an* application *of this operator.* □

However, there are two reasons why this simplistic approach will not work. First of all, EMFCompare documentation states that not all difference kinds are allowed for particular structural features [40]. Second, the datatype ν^* corresponding to ϵ might be abstract, *i.e.,*there might be no values corresponding to it. In the remainder of this section we discuss how these two issues are addressed. Details of the operator generation process are available in the technical report [41].

Firstly, depending on the structural feature ϵ, EMFCompare restricts difference kinds δ as follows.

If the feature ϵ is a containment reference , the change kind can only be `ADD` or `DELETE`, as in Example III.3.

Example III.3. *The operator corresponding to adding a class is* $\langle$`ADD`, `EPackage`, *eClassifiers*, `EClass`$\rangle$. *This is because eClassifiers is a containment reference (*i.e., *classes are contained in a package).*

The operator $\langle$`CHANGE`, `EPackage`, *eClassifiers*, `EClass`$\rangle$ *is thus excluded by EMFCompare constraints.* □

If ϵ is not a containment reference and the upper bound of the feature is greater than one, the δ can only be `ADD` or `DELETE`. If the upper bound of the feature is one, δ can only be `CHANGE`, as in Example III.4.

[5]EMFCompare also support the `MOVE` change kind for re-ordering elements. For simplicity, we omit changes of this kind.

Example III.4. *The operator for changing the eOpposite relation (which has an upper bound of one), would be:* $\langle$`CHANGE`, `EReference`, *eOpposite*, `EReference`$\rangle$

*Whereas the operator for adding a class to the eSuperTypes relation (which has an upper bound of *), would be:* $\langle$`ADD`, `EClass`, *eSuperTypes*, `EClass`$\rangle$ □

As mentioned above, the second challenge pertaining to operator generation is related to the datatype of ϵ being abstract, meaning one can never have an instance of it. To mitigate this, we create a separate operator for every possible type that is not *abstract* (*i.e.,* its `abstract` feature is false) and that is derivable from the type of ϵ (note that this may include the type of ϵ itself).

Example III.5. *When adding a class to a package we observe that the type of the eClassifiers feature is* `EClassifier`. *However, as shown on Fig. 2* `EClassifier` *is abstract and has two sub-classes* `EClass` *and* `EDataType`[6]. *Those sub-classes are not abstract. Hence, instead of creating a single operator* $\langle$`ADD`, `EPackage`, *eClassifiers*, `EClassifier`$\rangle$, *we create* $\langle$`ADD`, `EPackage`, *eClassifiers*, `EClass`$\rangle$ *and* $\langle$`ADD`, `EPackage`, *eClassifiers*, `EDataType`$\rangle$. □

The library obtained as explained above constitutes the complete library of atomic operators $\mathcal{R}$, presented in Table V. Using this library, every change on `Ecore`-based meta-models can be expressed. Observe that a similar approach could have been applied to a non-`Ecore` meta-meta-model, yielding a variant on $\mathcal{R}$ specific to that meta-meta-model.

C. Getting the Evolution History for the Case Study

As was stated at the beginning of Section III, it is necessary to reconstruct the *evolution history* of the case study in terms of $\mathcal{R}$. To do so, EMFCompare [39] is run on subsequent version pairs of each of the 22 DSLs in the DSL ecosystem. We obtain the subsequent versions of the twenty-two DSLs from a subversion repository. The ecosystem development team employs a trunk-branch development scheme. We have extracted subsequent version pairs from the trunk only, as the DSLs in trunk represent stable releases.

The difference models yielded by running EMFCompare on the subsequent versions contain a number of atomic differences. We refer to this collection of differences as the *evolution history*.

Correctness of our *evolution history* is threatened by the accuracy of EMFCompare results, as recognized by Herrmannsdörfer *et al.* [32]. To assess to what extent this is a real threat to validity, the evolution of two DSLs in the ecosystem was reconstructed manually by a graduate student [42], [43], who concluded that the results of EMFCompare were perfect with the exception of detecting renames. To address the rename detection, a more structural approach to model differencing as described by Protić [44] could be considered. We consider this as future work.

[6]In the actual `Ecore` meta-meta-model has additional sub-classes.

Each difference in the *evolution history* can be seen as the result of an operator application. To reconstruct this relation, each difference in the *evolution history* is related to an atomic operator in $\mathcal{R}$. Note that this is straightforward as we constructed each of these atomic operators based on the EMFCompare difference model.

D. Relating the Operator Library to $\mathcal{R}$

Lastly, we manually relate each atomic operator from $\mathcal{H}$ to an atomic operator in our computed set $\mathcal{R}$, as was illustrated in Fig. 5.

Note, there might be operators in $\mathcal{R}$ that are not related to any operator in the library under study. For example in Fig. 5, there is no operator h_m in the library under study such that h_m is related to r_5. This means that r_5 is not supported by the library under study. Additionally, as illustrated in Fig. 5, there may be one-to-many, and many-to-one relations between the operators in $\mathcal{R}$ and operators in $\mathcal{H}$.

IV. Results

A. Complete Library of Atomic operators

Having computed $\mathcal{R}$ as described in Section III-B, we have obtained a complete library of atomic operators (for `Ecore`-based meta-models). This library consists of 176 atomic operators and is presented in Table V.

B. The CARM Case Study

Applying EMFCompare to the subsequent version pairs of the CARM case study results in a total of 3405 individual differences that constitute our *evolution history*. Subsequently relating these differences to operators in $\mathcal{R}$ reveals that these 3405 differences can be specified using applications of 70 distinct atomic operators (please recall from Section III-B that these numbers exclude `MOVE` operations). In Fig. 6 the number of applications per operator is illustrated (note that the operator names have been omitted for readability). In this distribution, positive skew can be observed. Further analysis shows that the first thirteen operators (*i.e.,* the thirteen operators with the highest number of applications) together are related to 2756 of the differences yielded by EMFCompare. This means that the first thirteen operators specify 81% of the *evolution history* of our case study.

The differences yielded by EMFCompare with respect the CARM use case are summarized in Table I.

C. $\mathcal{H}$: the Library of Herrmannsdörfer et al.

The library of Herrmannsdörfer *et al.* [14], $\mathcal{H}$, consists of 61 operators, of which 30 are atomic, and 31 are compound. After having manually related the atomic operators in $\mathcal{H}$ to the atomic operators in $\mathcal{R}$, we observe that 41 of the atomic operators in $\mathcal{R}$ are covered by the atomic operators in $\mathcal{H}$.

This discrepancy is caused by many-to-one, and one-to-many relations between both libraries. For example, the operators `RenameAttribute` and `RenameClass` in $\mathcal{R}$ are related to the `Rename` operator in $\mathcal{H}$. On the other hand, the operator `Change Class Abstract` in $\mathcal{R}$ is related to two operators: `Make Class Abstract` and `Drop Class Abstract`, in $\mathcal{H}$.

TABLE I: A brief summary of the CARM *evolution history* yielded by EMFCompare. The number of differences has been presented per meta-meta-model concept and per difference kind.

Meta-meta-model concept / Change kind	ADD	CHANGE	DELETE	Grand Total
EAnnotation	22	0	34	56
EAttribute	0	462	0	462
EClass	811	68	510	1389
EDataType	0	0	0	0
EEnum	107	2	37	146
EEnumLiteral	0	23	0	23
EFactory	0	0	0	0
EGenericType	7	15	0	22
EOperation	41	34	34	109
EPackage	233	32	131	396
EParameter	7	30	0	37
EReference	38	712	13	763
EStringToString MapEntry	0	2	0	2
ETypeParameter	0	0	0	0
Grand Total	1266	1380	759	3405

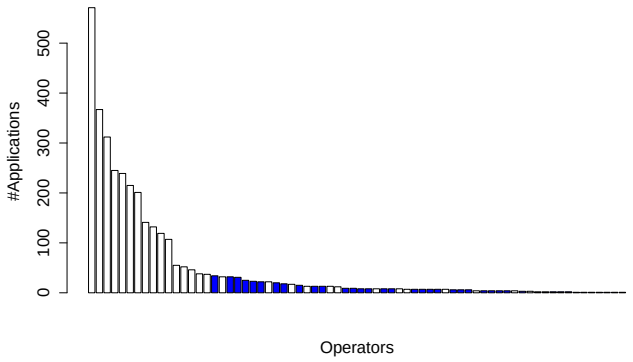


Fig. 6: A barplot showing the number of applications (y-axis) of each operator in $\mathcal{R}$ (x-axis) to the CARM case study. Bars corresponding to operators not in $\mathcal{H}$ have been colored blue. The distribution has a median of 8, and the Gini-index [45] is 0.75 of the maximal 0.98 [46].

D. Answering *RQs*

In this section we will discuss how the differences in the *evolution history*, $\mathcal{H}$, and $\mathcal{R}$ relate. The results discussed are are illustrated in Fig. 7.

To answer **RQ1**, we have related the 3405 differences in the CARM case study to 70 operators in $\mathcal{R}$, and the operators in $\mathcal{H}$ to 41 operators in $\mathcal{R}$. Using these relations we observe that $\mathcal{H}$ supports 32 of the 70 operators used by the CARM case study. In total, these 32 operators supported by $\mathcal{H}$ are able to describe 3032 of the 3405 (89%) of the differences in the CARM *evolution history*.

Next, to answer **RQ2**, we look at operators in $\mathcal{R}$ that are not related to operators in $\mathcal{H}$. In Fig. 7 it can be observed that there are 135 deficiencies ($= 176 - 41$) in $\mathcal{H}$. For example, changing the `isUnique` field of an attribute, or changing the `isOrdered` field of an attribute, are among

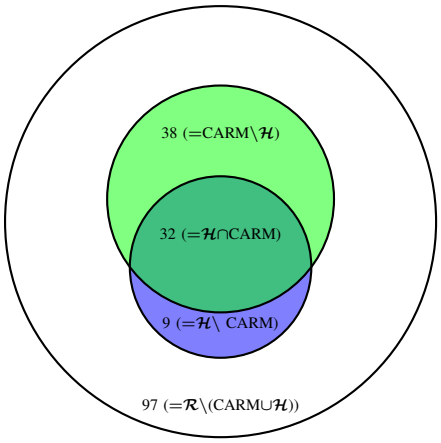


Fig. 7: An illustration of how the operators offered by the library of Herrmannsdörfer *et al.* [14] $\mathcal{H}$ (blue), and the CARM case study (green) relate to the complete library of 176 atomic operators $\mathcal{R}$ (white).

TABLE II: A subset of Table V containing an overview of the thirteen most frequently used operators of the CARM use-case, number of applications, percentage of the *evolution history* and support by $\mathcal{H}$.

Operation	Value type	applications in CARM	$\mathcal{H}$
`CHANGE` reference type	`EClass`	571 (16.7%)	Yes
`CHANGE` attribute type	`EDataType`	367 (10.7%)	Yes
`ADDNEW` reference to class	`EReference`	312 (9.2%)	Yes
`ADDNEW` attribute to class	`EAttribute`	245 (7.2%)	Yes
`ADD` supertype	`EClass`	239 (7.0%)	Yes
`ADDNEW` class to package	`EClass`	215 (6.3%)	Yes
`DELETE` reference from class	`EReference`	201 (5.9%)	Yes
`DELETE` attribute from class	`EAttribute`	141 (4.1%)	Yes
`DELETE` supertype	`EClass`	132 (3.9%)	Yes
`DELETE` class from package	`EClass`	119 (3.5%)	Yes
`ADDNEW` literal to an enumeration	`EEnumLiteral`	107 (3.1%)	Yes
`CHANGE` name of a class	`EString`	55 (1.6%)	Yes
`CHANGE` name of a reference	`EString`	52 (1.5%)	Yes
		2756 (81)%	

these deficiencies.

Finally, to answer **RQ3**, we look at operators in $\mathcal{R}$ that are neither related to differences in the CARM case study, nor to operators in $\mathcal{H}$. In Fig. 7, observe that of the 135 deficiencies in $\mathcal{H}$, 97 ($= 176 - 38 - 32 - 9$) are not revealed by the CARM case study (*i.e.,* 72%). For example, the deficiencies mentioned in the results of **RQ2**, are not revealed by the CARM case study.

V. Discussion

A. Contribution to *RQ1*

The first question we asked was "To what extent can existing operator libraries specify evolution of DSLs in a large-scale industrial MDSE ecosystem?". We answer this by investigating what portion of the *evolution history* of our industrial case study could be covered by the library of Herrmannsdörfer *et al.* [14] ($\mathcal{H}$). In Section IV we have seen that $\mathcal{H}$ provides support for nine atomic operators that are not required by the CARM case study. Furthermore, 32 atomic operators are offered by $\mathcal{H}$ that are required by the CARM case study. However, the CARM case study requires support for 70. We conclude that 54% ($= 38/70$) of operators required for the case study are not provided by $\mathcal{H}$.

However, the number of applications among these operators is unevenly distributed in our case-study. As can be observed in Fig. 6, the number of applications per operator shows positive skew. Further inspection shows that the Pareto principle (*cf.* [47]) applies: 81% of the *evolution history* can be specified using 19% ($= 13/70$) of the 70 operators required (*cf.* Table II).

In Fig. 6, bars corresponding to operators not supported by $\mathcal{H}$ have been colored. Although the most frequently required operators are supported (*i.e.,* colored white), the colored operators in the tail of the distribution seem arbitrary. Further inspection of these colored operators in Fig. 6 reveals that they belong to a particular class of operators.

In a previous case study [32] Herrmannsdörfer *et al.* have classified evolution operators with respect to the automatability of their coupled (co-evolution) operators. The resulting classification is illustrated in Fig. 8. The classification distinguishes [32]:

1) **meta-model only operators** for which no model co-evolution is required;
2) **meta-model independent operators** for which the model co-evolution is independent of a particular meta-model;
3) **meta-model specific operators** for which the model co-evolution depends on the structure of the meta-model being evolved;
4) **model-specific operators** for which the model co-evolution depends on the structure of the model being co-evolved.

Herrmannsdörfer *et al.* conclude that for each successive class, the automatability decreases (as illustrated in Fig. 8). We observe that most of the colored operators in Fig. 6 correspond to the classes with lower automatability: meta-model specific and model-specific. The results for **RQ1** (in Section IV-D), the operators in $\mathcal{H}$ cannot specify 11% of the *evolution history*, is comparable to the earlier study by Herrmannsdörfer *et al.* [32], where deficits of 7% and 15% are observed.

Detailed analyses of the differences supported and not supported by $\mathcal{H}$ respectively are presented in Table III. The most prominent differences that are not supported are related to the `EAnnotation` (56 differences) and `EOperation` (108 differences) meta-meta-model concepts. We further note the absence of `ADD` operators for `EReference` features, and `DELETE` operators for `EClass` features.

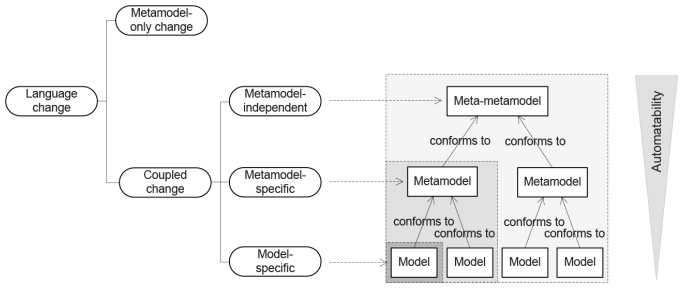


Fig. 8: Classification of meta-model adaption, and the automatability of their co-evolution for models [32].

TABLE III: Summary of CARM differences are supported/not supported by the library of Herrmannsdörfer *et al.* [14] ($\mathcal{H}$) respectively.

Meta-meta-model element / change kind	ADD	CHANGE	DELETE	Grand Total
EAnnotation	0/22	0/0	0/34	0/**56**
EAttribute	0/0	459/3	0/0	459/3
EClass	796/15	67/1	474/**36**	1337/52
EDataType	0/0	0/0	0/0	0/0
EEnum	107/0	2/0	37/0	146/0
EEnumLiteral	0/0	4/19	0/0	4/19
EFactory	0/0	0/0	0/0	0/0
EGenericType	0/7	0/15	0/0	0/22
EOperation	0/41	1/33	0/34	1/**108**
EPackage	232/1	7/25	130/1	369/27
EParameter	0/7	4/26	0/0	4/33
EReference	0/**38**	712/0	0/13	712/51
EString-ToString-MapEntry	0/0	0/2	0/0	0/2
ETypeParameter	0/0	0/0	0/0	0/0
Grand Total	1135/131	1256/124	641/118	3032/373

B. Contribution to *RQ2*

To answer **RQ2**, "TWhich operators from $\mathcal{R}$ are missing form state-of-the-art operator libraries?", we compared $\mathcal{H}$ to $\mathcal{R}$ (Table V).

The library $\mathcal{H}$, covers 41 of the atomic operators in $\mathcal{R}$. Although in Section IV-C we have observed that these 41 operators are sufficient to cover 89% of the *evolution history* of our industrial case study, we also observed that there are 38 kinds of differences in the case study not covered by the library. Comparing $\mathcal{H}$ to $\mathcal{R}$ shows that a total amount of 135 operators lack supported. Further analysis of these 135 deficiencies reveals that several Ecore concepts are under-supported or not supported at all. For instance, EOperators are insufficiently supported, and no operators for EAnnotations exist at all. The latter is even more surprising since EAnnotation has been reported to be one of the most widely used Ecore classes [48]. If operators for EOperators and EAnnotations would be added, an additional 5% of the *evolution history* would be supported (for a total of 94% coverage).

Summarizing, although evolution of DSLs in an industrial context can be specified to a large extent using $\mathcal{H}$, the library lacks expressiveness for evolution specification in general.

TABLE IV: The number of operators that would not have been discovered using our industrial case-study, per meta-meta-concept.

Meta-meta-model concept	Number of Operators not discovered
EAnnotation	19
EAttribute	11
EClass	4
EDataType	7
EEnum	7
EEnumLiteral	3
EFactory	2
EGenericType	7
EOperation	14
EPackage	1
EParameter	6
EReference	11
EStringToStringMapEntry	1
ETypeParameter	4

An additional 135 atomic operators would need to be added to the library in order to resolve this deficiency.

C. Contributions to *RQ3*

To answer **RQ3** "What are the limitations of our case study with respect to finding operator library deficiencies?" we compare the results of **RQ1** and **RQ2**.

Of the 135 deficiencies in $\mathcal{H}$, only 38 were revealed using our case study. This leaves a total of 97 of 135 deficiencies (72%). In Table IV, the precise number of missing operators per meta-meta-model element are presented. It can be observed that the large number of missing operators is mainly caused by EAnnotations and EOperations. This is either due to our case study lacking these meta-meta-concepts, or existing meta-meta-concepts not evolving. However, to be sure, more research is needed.

VI. Research Limitations

Several threats might affect validity of our results.

Firstly, we compute the *evolution history* using EMFCompare. The accuracy of EMFCompare may have affected the results in **RQ1**. To mitigate this threat, the EMFCompare results have been manually verified by a graduate student. Furthermore, the relation between the library of Herrmannsdörfer *et al.* [14] and $\mathcal{R}$, was performed manually, and is thus susceptible to human error. To mitigate this error, the created relation was verified by the same graduate student.

Furthermore, we obtain the subsequent versions of DSLs from a subversion (SVN) version-control system. Using SVN, the CARM development team employs a trunk-branch development scheme. We have studied the subsequent DSL versions in the trunk, as these are finished products, rather than unstable development versions. This means that we might miss changes that only occur in development branches.

The validity of our results in **RQ1** might also be influenced by the choice of our case study. Although our case study is large, there is no guarantee that our results can be generalized to other industrial DSL ecosystems. To mitigate this, in **RQ2**, we have presented a theoretically complete library of atomic operators that is independent of any case study, and in **RQ3** we have studied the actual limitations of our case study.

VII. Conclusions

This paper evaluated the ability of the atomic operators in the state-of-the-art library of Herrmannsdörfer *et al.*[14] to specify evolution of a large-scale industrial case study of twenty-two DSLs. We conclude that this library can specify 89% of DSL evolutions, and we have identified the deficiencies that make up the remaining 11%. Next, we have presented a top-down methodology for creating a theoretically complete library of atomic operators. Although we have only calculated this library for the `Ecore` meta-meta-model, our approach is generic and may be applied to any meta-meta-model. Using this library (Table V), we have revealed further deficiencies in existing operator libraries, and have identified the operators that would need to be implemented to mitigate these deficiencies.

As *future work*, we consider including compound operators. Furthermore, we wish to formalize our library of atomic operators ($\mathcal{R}$) into an executable DSL for meta-model evolution, and extend this DSL with support for compound operators. After having formalized the evolution DSL, we plan to study co-evolution specifications and their relations to evolution specifications. Finally, as mentioned above, we plan to replicate the current study using an approach advocated by Protić [44] as an alternative to EMFCompare.

Acknowledgements

This research is funded in whole by ASML. We would like to thank the CARM development team for their cooperation and Yorrick Visssers for his effort in validating EMFCompare for our data set.

TABLE V: Operator library $\mathcal{R}$ partitioned by operator kind (ADD, CHANGE, DELETE). Each entry represents an operator, *e.g.,* "**ADD** an **EObject** to the **contents** feature of an **EAnnotation**".

ParentElement	StructuralFeature	ValueType
	ADD	
EAnnotation	contents	EObject
EAnnotation	details	EStringToStringMapEntry
EAnnotation	eAnnotations	EAnnotation
EAnnotation	references	EObject
EAttribute	eAnnotations	EAnnotation
EAttribute	eGenericType	EGenericType
EClass	eAnnotations	EAnnotation
EClass	eGenericSuperTypes	EGenericType
EClass	eOperations	EOperation
EClass	eStructuralFeatures	EAttribute
EClass	eStructuralFeatures	EReference
EClass	eSuperTypes	EClass
EClass	eTypeParameters	ETypeParameter
EDataType	eAnnotations	EAnnotation
EDataType	eTypeParameters	ETypeParameter
EEnumLiteral	eAnnotations	EAnnotation
EEnum	eAnnotations	EAnnotation
EEnum	eLiterals	EEnumLiteral
EEnum	eTypeParameters	ETypeParameter
EFactory	eAnnotations	EAnnotation
EGenericType	eLowerBound	EGenericType
EGenericType	eTypeArguments	EGenericType
EGenericType	eUpperBound	EGenericType
EOperation	eAnnotations	EAnnotation
EOperation	eExceptions	EClass
EOperation	eExceptions	EDataType
EOperation	eExceptions	EEnum
EOperation	eGenericExceptions	EGenericType
EOperation	eGenericType	EGenericType
EOperation	eParameters	EParameter
EOperation	eTypeParameters	ETypeParameter
EPackage	eAnnotations	EAnnotation
EPackage	eClassifiers	EClass
EPackage	eClassifiers	EDataType
EPackage	eClassifiers	EEnum
EPackage	eSubpackages	EPackage
EParameter	eAnnotations	EAnnotation
EParameter	eGenericType	EGenericType
EReference	eAnnotations	EAnnotation
EReference	eGenericType	EGenericType
EReference	eKeys	EAttribute
ETypeParameter	eAnnotations	EAnnotation
ETypeParameter	eBounds	EGenericType
	CHANGE	
EAnnotation	eModelElement	EAnnotation
EAnnotation	eModelElement	EAttribute
EAnnotation	eModelElement	EClass
EAnnotation	eModelElement	EDataType
EAnnotation	eModelElement	EEnum
EAnnotation	eModelElement	EEnumLiteral
EAnnotation	eModelElement	EFactory
EAnnotation	eModelElement	EOperation
EAnnotation	eModelElement	EPackage
EAnnotation	eModelElement	EParameter
EAnnotation	eModelElement	EReference
EAnnotation	eModelElement	ETypeParameter
EAnnotation	source	String
EAttribute	changeable	boolean
EAttribute	defaultValueLiteral	String
EAttribute	derived	boolean
EAttribute	eType	EClass
EAttribute	eType	EDataType
EAttribute	eType	EEnum
EAttribute	iD	boolean
EAttribute	lowerBound	int
EAttribute	name	String
EAttribute	ordered	boolean
EAttribute	transient	boolean
EAttribute	unique	boolean
EAttribute	unsettable	boolean
EAttribute	upperBound	int
EAttribute	volatile	boolean
EClass	abstract	boolean
EClass	instanceClassName	String
EClass	instanceTypeName	String
EClass	interface	boolean
EClass	name	String
EDataType	instanceClassName	String
EDataType	instanceTypeName	String
EDataType	name	String
EDataType	serializable	boolean
EEnumLiteral	instance	EEnumerator
EEnumLiteral	literal	String
EEnumLiteral	name	String
EEnumLiteral	value	int
EEnum	instanceClassName	String
EEnum	instanceTypeName	String
EEnum	name	String
EEnum	serializable	boolean

ParentElement	StructuralFeature	ValueType
	CHANGE	
EFactory	ePackage	EPackage
EGenericType	eClassifier	EClass
EGenericType	eClassifier	EDataType
EGenericType	eClassifier	EEnum
EGenericType	eTypeParameter	ETypeParameter
EOperation	eType	EClass
EOperation	eType	EDataType
EOperation	eType	EEnum
EOperation	lowerBound	int
EOperation	name	String
EOperation	ordered	boolean
EOperation	unique	boolean
EOperation	upperBound	int
EPackage	eFactoryInstance	EFactory
EPackage	name	String
EPackage	nsPrefix	String
EPackage	nsURI	String
EParameter	eType	EClass
EParameter	eType	EDataType
EParameter	eType	EEnum
EParameter	lowerBound	int
EParameter	name	String
EParameter	ordered	boolean
EParameter	unique	boolean
EParameter	upperBound	int
EReference	changeable	boolean
EReference	containment	boolean
EReference	defaultValueLiteral	String
EReference	derived	boolean
EReference	eOpposite	EReference
EReference	eType	EClass
EReference	eType	EDataType
EReference	eType	EEnum
EReference	lowerBound	int
EReference	name	String
EReference	ordered	boolean
EReference	resolveProxies	boolean
EReference	transient	boolean
EReference	unique	boolean
EReference	unsettable	boolean
EReference	upperBound	int
EReference	volatile	boolean
EStringToStringMapEntry	key	String
EStringToStringMapEntry	value	String
ETypeParameter	name	String
	DELETE	
EAnnotation	contents	EObject
EAnnotation	details	EStringToStringMapEntry
EAnnotation	eAnnotations	EAnnotation
EAnnotation	references	EObject
EAttribute	eAnnotations	EAnnotation
EAttribute	eGenericType	EGenericType
EClass	eAnnotations	EAnnotation
EClass	eGenericSuperTypes	EGenericType
EClass	eOperations	EOperation
EClass	eStructuralFeatures	EAttribute
EClass	eStructuralFeatures	EReference
EClass	eSuperTypes	EClass
EClass	eTypeParameters	ETypeParameter
EDataType	eAnnotations	EAnnotation
EDataType	eTypeParameters	ETypeParameter
EEnumLiteral	eAnnotations	EAnnotation
EEnum	eAnnotations	EAnnotation
EEnum	eLiterals	EEnumLiteral
EEnum	eTypeParameters	ETypeParameter
EFactory	eAnnotations	EAnnotation
EGenericType	eLowerBound	EGenericType
EGenericType	eTypeArguments	EGenericType
EGenericType	eUpperBound	EGenericType
EOperation	eAnnotations	EAnnotation
EOperation	eExceptions	EClass
EOperation	eExceptions	EDataType
EOperation	eExceptions	EEnum
EOperation	eGenericExceptions	EGenericType
EOperation	eGenericType	EGenericType
EOperation	eParameters	EParameter
EOperation	eTypeParameters	ETypeParameter
EPackage	eAnnotations	EAnnotation
EPackage	eClassifiers	EClass
EPackage	eClassifiers	EDataType
EPackage	eClassifiers	EEnum
EPackage	eSubpackages	EPackage
EParameter	eAnnotations	EAnnotation
EParameter	eGenericType	EGenericType
EReference	eAnnotations	EAnnotation
EReference	eGenericType	EGenericType
EReference	eKeys	EAttribute
ETypeParameter	eAnnotations	EAnnotation
ETypeParameter	eBounds	EGenericType

REFERENCES

[1] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[2] J.-M. Favre, "Languages evolve too! changing the software time scale," in *Principles of Software Evolution*, 2005, pp. 33–42.

[3] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Coupled evolution in model-driven engineering," *IEEE Software*, vol. 29, no. 6, pp. 78–84, 2012.

[4] B. Gruschko, D. Kolovos, and R. Paige, "Towards synchronizing models with evolving metamodels," in *Workshop on Model-Driven Software Evolution*, 2007.

[5] A. Narayanan, T. Levendovszky, D. Balasubramanian, and G. Karsai, "Automatic domain model migration to manage metamodel evolution," in *MoDELS*, ser. LNCS. Springer, 2009, vol. 5795, pp. 706–711.

[6] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in *IEEE Enterprise Distributed Object Computing Conference*, 2008, pp. 222–231.

[7] G. Wachsmuth, "Metamodel adaptation and model co-adaptation," in *ECOOP*, ser. LNCS. Springer, 2007, vol. 4609, pp. 600–624.

[8] J. García, O. Diaz, and M. Azanza, "Model transformation co-evolution: A semi-automatic approach," in *SLE*, ser. LNCS. Springer, 2013, vol. 7745, pp. 144–163.

[9] T. Levendovszky, D. Balasubramanian, A. Narayanan, and G. Karsai, "A novel approach to semi-automated evolution of dsml model transformation," in *SLE*, ser. LNCS. Springer, 2010, vol. 5969, pp. 23–41.

[10] D. Di Ruscio, R. Lämmel, and A. Pierantonio, "Automated co-evolution of GMF editor models," in *SLE*, ser. LNCS. Springer, 2011, vol. 6563, pp. 143–162.

[11] B. Meyers and H. Vangheluwe, "A framework for evolution of modelling languages," *Science of Computer Programming*, vol. 76, no. 12, pp. 1223–1246, 2011.

[12] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," in *ICSE*, 2008, pp. 511–520.

[13] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack, "An analysis of approaches to model migration," in *MoDSE-MCCM Workshop*, 2009, pp. 6–15.

[14] M. Herrmannsdörfer, S. D. Vermolen, and G. Wachsmuth, "An extensive catalog of operators for the coupled evolution of metamodels and models," in *SLE*, ser. LNCS. Springer, 2011, vol. 6563, pp. 163–182.

[15] "Ecore," lhttp://www.eclipse.org/modeling/emf/, accessed: 2016-7-20.

[16] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley, 2009.

[17] J. Bézivin, "On the unification power of models," *Software & Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[18] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.

[19] J. Bézivin, "Model driven engineering: An emerging technical space," in *Generative and Transformational Techniques in Software Engineering*, ser. LNCS. Springer, 2006, vol. 4143, pp. 36–64.

[20] "Eclipse," http://www.eclipse.org/, accessed: 2015-04-07.

[21] "Eclipse Juno Documentation," http://help.eclipse.org/juno/index.jsp, accessed: 2015-10-07.

[22] A. Cicchetti, D. Di Ruscio, and A. Pierantonio, "A metamodel independent approach to difference representation," *Journal of Object Technology*, vol. 6, no. 9, pp. 165–185, 2007.

[23] D. Ohst, M. Welle, and U. Kelter, "Differences between versions of UML diagrams," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 5, pp. 227–236, 2003.

[24] T. Mens, "A state-of-the-art survey on software merging," *IEEE Transactions on Software Engineering*, vol. 28, no. 5, pp. 449–462, 2002.

[25] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. Polack, "Model migration with Epsilon Flock," in *Theory and Practice of Model Transformations*, ser. LNCS. Springer, 2010, vol. 6142, pp. 184–198.

[26] M. Herrmannsdörfer and G. Wachsmuth, "Coupled evolution of software metamodels and models," in *Evolving Software Systems*. Springer, 2014, pp. 33–63.

[27] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer, "Model transformation by-example: A survey of the first wave," in *Conceptual Modelling and Its Theoretical Foundations*, ser. LNCS. Springer, 2012, vol. 7260, pp. 197–215.

[28] G. Taentzer, F. Mantz, and Y. Lamo, "Co-transformation of graphs and type graphs with application to model co-evolution," in *International Conference on Graph Transformations*. Springer, 2012, pp. 326–340.

[29] F. Mantz, G. Taentzer, Y. Lamo, and U. Wolter, "Co-evolving meta-models and their instance models: A formal approach based on graph transformation," *Science of Computer Programming*, vol. 104, pp. 2–43, 2015.

[30] M. Herrmannsdörfer, "COPE - A workbench for the coupled evolution of metamodels and models," in *SLE*, ser. LNCS. Springer, 2011, vol. 6563, pp. 286–295.

[31] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Software Engineering*, vol. 13, pp. 211–218, 2008.

[32] M. Herrmannsdörfer, S. Benz, and E. Juergens, "Automatability of coupled evolution of metamodels and models in practice," in *MoDELS*, ser. LNCS. Springer, 2008, vol. 5301, pp. 645–659.

[33] H. Zhong and Z. Su, "An empirical study on real bug fixes," in *ICSE*. IEEE, 2015, pp. 913–923.

[34] A. Serebrenik and T. Mens, "Challenges in software ecosystems research," in *Software Architecture Workshops*, I. Crnkovic, Ed. ACM, 2015, pp. 40:1–40:6.

[35] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik, "Studying evolving software ecosystems based on ecological models," in *Evolving Software Systems*. Springer, 2014, pp. 297–326.

[36] R. R. H. Schiffelers, W. Alberts, and J. P. M. Voeten, "Model-based specification, analysis and synthesis of servo controllers for lithoscanners," in *6th International Workshop on Multi-Paradigm Modeling*. ACM, 2012, pp. 55–60.

[37] C. M. Gerpheide, R. R. H. Schiffelers, and A. Serebrenik, "Assessing and improving quality of QVTo model transformations," *Software Quality Journal*, vol. 24, no. 3, pp. 797–834, 2016.

[38] "ASML," http://www.asml.com/, accessed: 2015-04-07.

[39] "EMF Compare," https://www.eclipse.org/emf/compare/, accessed: 2015-04-07.

[40] "EMFCompare developer guide," https://www.eclipse.org/emf/compare/documentation/latest/developer/developer-guide.html, accessed: 2015-10-06.

[41] J.G.M. Mengerink, R.R.H. Schiffelers, A. Serebrenik, and M.G.J. van den Brand, "Evolution specification evaluation in industrial mdse ecosystems," Eindhoven University of Technology, Tech. Rep. CSR-15-04, 2015. [Online]. Available: https://pure.tue.nl/ws/files/3757969/390954927658277.pdf

[42] Y. Vissers, "Using Edapt for Coupled Evolution of Metamodels and Models," Master's thesis, Eindhoven University of Technology, the Netherlands, 2015.

[43] Y. Vissers, J. G. M. Mengerink, R. R. H. Schiffelers, A. Serebrenik, and M. Reniers, "Maintenance of specification models in industry using edapt," in *FDL*, 2016.

[44] Z. Protić, "Configuration management for models: Generic methods for model comparison and model co-evolution," Ph.D. dissertation, Eindhoven University of Technology, 2011.

[45] C. Gini, "Measurement of inequality of incomes," *The Economic Journal*, vol. 31, pp. 124–126, 1921.

[46] P. D. Allison, "Measures of inequality," *American Sociological Review*, vol. 43, no. 6, pp. 865–880, 1978.

[47] M. Goeminne and T. Mens, "Evidence for the pareto principle in open source software activity," in *Joint Proceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, ser. CEUR-WS, vol. 708, 2011, pp. 74–82.

[48] M. Herrmannsdoerfer, D. Ratiu, and M. Koegel, *Metamodel Usage Analysis for Identifying Metamodel Improvements*. Springer, 2011, pp. 62–81.