Evolution specification evaluation in industrial MDSE ecosystems

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

Evolution Specification Evaluation in Industrial MDSE Ecosystems

J.G.M. Mengerink, R.R.H. Schiffelers, A. Serebrenik, M.G.J. van den Brand

15/04

ISSN 0926-4515

editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 15-04
Eindhoven, October 2015

Evolution Specification Evaluation in Industrial MDSE Ecosystems

Josh G.M. Mengerink*, Ramon R.H. Schiffelers*†, Alexander Serebrenik*, Mark G.J. van den Brand*
* Eindhoven University of Technology, Eindhoven, The Netherlands
Email:{j.g.m.mengerink, a.serebrenik, m.g.j.v.d.brand}@tue.nl
†ASML, Veldhoven, The Netherlands
Email: ramon.schiffelers@asml.com

***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. When the number of models that needs to co-evolve increases, so does the required effort to do so. This is called the *co-evolution problem*.**

We have investigated the extent of the co-evolution problem at ASML [1], provider of lithography equipment for the semiconductor industry. Here we have described the structure and evolution of a large-scale ecosystem of DSLs. We have observed that due to the large number of artifacts that require co-evolutionary activity, manual solutions have become unfeasible, and an automated approach is required. A popular approach for automating co-evolution is the operator-based approach. In this paper we have evaluated the operator-based approach on a large-scale industrial case-study of twenty-two DSLs and 95 model-to-model transformations with a revision history of over three years, and have revealed deficiencies in existing operator libraries. To address these deficiencies we have presented a top-down methodology to derive a complete set of operators.

I. Introduction

Domain-specific languages (DSLs) offer an efficient way to model complex systems in terms of familiar domain concepts. The relative ease with which model driven engineering (MDE) allows new DSLs to be created is allowing DSLs to be adopted more quickly in industry [2]. The amount of industrial case-studies in literature suggests that MDE is being adopted in industry [3], [4]. For example at ASML, where DSLs are being used for specification of servo-control applications and execution platforms for the TWINSCAN lithography machine [5].

DSLs are created by specifying their abstract syntax using meta-models[1] [6] that dictate the concepts and structure of a language. Due to this centralized specification of language concepts and structure, meta-models have become, by design, a hotspot in the development process: artifacts such as models, model-transformations, editors, and even Java code (*e.g.*, the EMF-API when using Eclipse EMF as an implementation technology) depend on the meta-models. This also means that when meta-models evolve, related artifacts such as models [7]–[10], model-transformations [11], [12], text editors, and graphical editors [13] must reflect changes made to the meta-models. This is known as the *co-evolution* problem [14].

Manual updating of models triggered by evolution of meta-models is not only difficult but also costly. One can imagine that, in some cases, the induced co-evolutionary effort that would be inferred by a meta-model change is so significant that the meta-model change would be postponed or even not executed at all. This causes developers to encode new domain concepts into existing ones, jeopardizing the primary strength of DSLs: modeling in a specific domain.

The contributions of this paper are threefold. First, we provide insights into how scale affects (co-)evolution in an ecosystem of DSLs used for systems engineering. We study how *evolution and co-evolution* are dealt with in a *large-scale industrial model-driven system engineering (MDSE) ecosystem* of twenty-two DSLs. Second, we investigate if an *operator-based approach is suitable* for dealing with evolution and co-evolution at this large scale. We investigate to what extent current operator libraries are able to specify (co-)evolution in our industrial case-study. Lastly, we investigate what extensions to operator libraries are *necessary* in order to *completely specify (co-)evolution* in practical use-cases. Rather than extending one of the existing operator libraries in an *ad hoc* way to fully cover co-evolution, we present a top-down methodology to derive a complete set of atomic operators. In this way, every meta-model evolution can be specified.

The remainder of this paper is structured as follows: In Section II we elaborate on MDE ecosystems and the added challenges they pose with respect to (co-)evolution. In Section III we explain the industrial context of our research, and in Section IV we look at the MDSE ecosystem case study in this industrial context. In Section V we analyze the application of the operator-based approach to our industrial case study, and identify shortcomings. We discuss how to mitigate these shortcomings in Section VI. In Section VII we discuss related work in our field. Finally, we conclude this paper by sketching directions for future work, and summarizing our conclusions in Section VIII.

II. (Co-)Evolution in MDSE Ecosystems

A. MDSE Ecosystems

In an MDSE ecosystem, meta-models are the central artifact, defining language concepts and structure. Many artifacts

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

depend on the meta-model for this structure. In Figure 1 an abstract representation of such an ecosystem has been illustrated. Here, two DSLs are illustrated (`DSLA` and `DSLB`), where a model-to-model transformation (`A2B.mtl`) maps models conforming to `DSLA` (*i.e.*, `alpha.DSLA`) to models conforming to `DSLB` (*i.e.*, `beta.DSLB`). Furthermore, domain concepts may be included between languages (*e.g.*, `DSLB` including domain concepts from `DSLA`). The meta-models specifying these DSLs are subject to change, for example: because of new insights into a domain, or technological advancement [15]. When these DSLs evolve (*e.g.*, `DSLA` to `DSLA'`) models conforming to these DSLs must co-evolve (*i.e.*, `alpha.DSLA` to `alpha'.DSLA'`). The process of determining how models must co-change, and actually co-changing them with respect to changing DSLs is referred to as the *co-evolution problem*.

B. Co-Evolution

With respect to model conformance [16], changes that make up the evolution of a meta-model may be classified into three different categories [7]:

1) **Non-breaking changes (NBC)**, which do not break model conformance;
2) **Breaking-Resolvable Changes (BRC)**, which break model conformance, but can be co-evolved in a fully automated fashion.
3) **Breaking-Unresolvable Changes (BUC)**, which break model conformance, and require additional information for successful co-evolution of models to take place.

The main challenges in model co-evolution arise with respect to BUCs, as they introduce ambiguities. For instance, in Figure 2 it is unclear whether `X.name` is renamed to `X'.id` or `Y'.name`, or if no rename (but adds and deletes) was performed. Due to the size of industrial use-cases (i.e. hundreds to thousands of model instances) manual resolution of these ambiguities is no longer feasible, and automation is required

A number of approaches have been proposed to address the co-evolution problem [17]. An approach that has been extensively studied is the operator-based approach [10], where a library of *co-evolutionary operators* is supplied to represent the modelers intent with respect to model evolution [18]. This approach is not only applicable to MDE, but has also been used in the context of databases [19]. Each operator specifies a meta-model evolution, along with a corresponding model co-evolution. For instance, the modification in Figure 2 could be described using the operators `Rename(X.name, "id")` and `CreateAttribute(Y, "name", EString)` [18]. The practical applicability of such an approach relies heavily on the set of available operators, such as those proposed by Herrmannsdörfer *et al.* [3], [4], [18].

When dealing with an MDSE ecosystem as mentioned in Section II, the magnitude of the co-evolution problem increases further. Evolution of a DSL may affect more artifacts than just the conforming models. This has been illustrated in the upper segment of Figure 1. When a DSL evolves (∂(`DSLA`)), its conforming models must co-evolve (∂(`alpha`)).

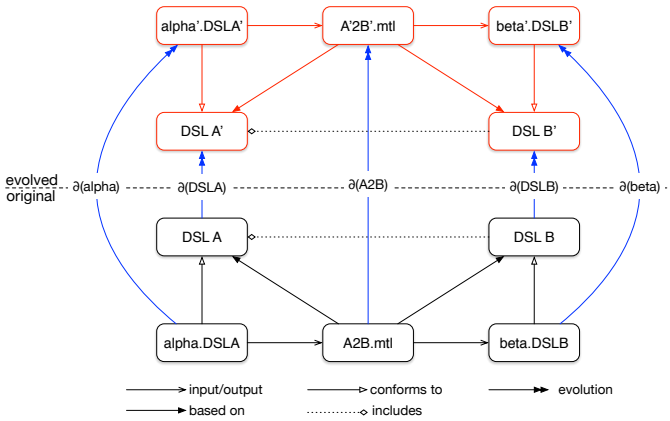


Fig. 1: Abstract representation of evolution in an MDSE ecosystem.

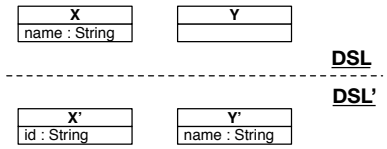


Fig. 2: Fragment of a meta-model: before (a) and after (b) the modification.

Additionally, model-to-model transformations might need adaptation (∂(`A2B`)) in order to deal with newly added concepts in the new models instances (`alpha'.DSLA'`). Lastly, `DSLB` might include domain concepts from `DSLA`. Just as models from `DSLA` need to co-evolve to capture the evolution in `DSLA`, models of `DSLB` including concepts from `DSLA` might also need to co-evolve (∂(`beta`)).

III. Industrial Context

Our research takes place at ASML [1], provider of lithography systems for the semiconductor industry. Over the last years, ASML uses model-driven engineering (MDE) in several parts of its development process.

ASMLs lithography systems consist of numerous servo control systems that are developed using MDE according to the Control Architecture Reference Model (CARM) [5]. CARM consists of multiple layers to describe the control logic and their execution platforms of the TWINSCAN lithoscanner at several levels of abstraction. CARM is multidisciplinary in nature, requiring expertise of mechatronic engineers, electrical engineers and embedded software engineers. Multiple abstraction layers as well as the multidisciplinary character of CARM necessitated development of a set of DSLs each targeting a different domain (cf. Figure 3). Dependencies between the DSLs, as well as the shared environment where they are developed and evolve, make the set of DSLs into a software ecosystem [20], [21].

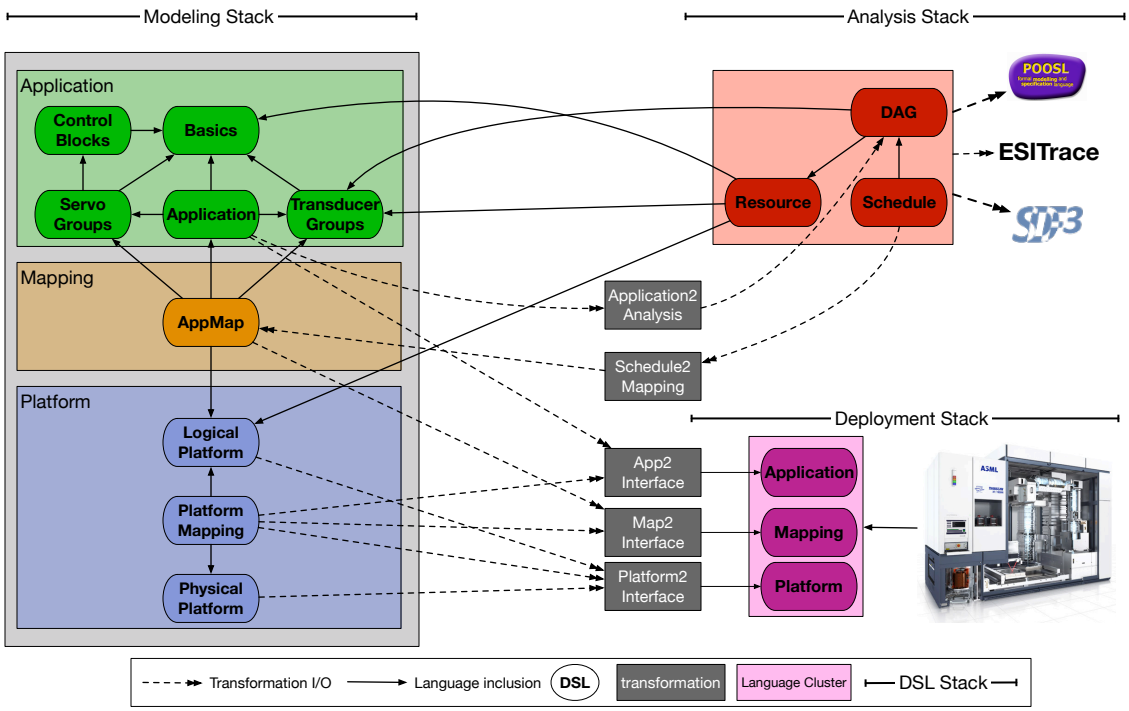


Fig. 3: A simplified illustration of ASMLs CARM MDSE ecosystem, where only the most important DSLs and model-transformations are shown.

The CARM ecosystem is divided into three main components: the modeling stack, the analysis stack and the deployment stack [5].

The **modeling stack** (Figure 3, left) allows for the specification of control applications for the litho-scanner at various levels of abstraction, and is modeled according to the Y-chart paradigm [22], which is a guideline for system decomposition. The *platform layer* (slate blue) has languages for modeling logical and physical properties of the platform. Finally, the *mapping* (orange) dictates how the *application* (green) is mapped to its execution platform.

The **analysis stack** (Figure 3, top right) allows for early predictability in the designs from the modeling stack, and plays a key role in scheduling the application onto the multi-processor platforms. Therefore, the analysis stack includes DSLs that can be analyzed by external verification and simulation tools such as POOSL [23], SDF [24], and ESITrace [25].

Finally, the **deployment stack** (Figure 3, bottom right) was designed to facilitate the disclosure of all relevant information to the clients on each TWINSCAN machine to initialize and execute the process controllers as well as configuring their execution platforms.

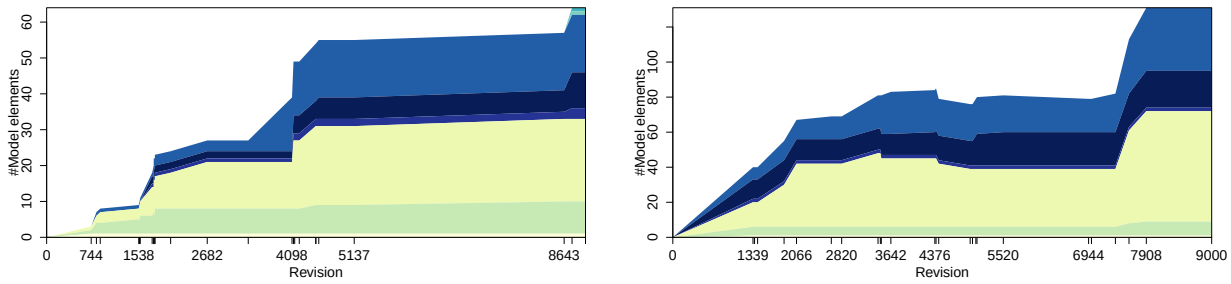
All DSLs in the CARM ecosystem are defined in terms of EMF Ecore class diagrams [26], [27] and OCL constraints [28]. The languages consist, on average, of 60 to 80 concepts with outliers on both the high end (400 concepts) and the low end (10 concepts). Some languages share common functionality, or contain strongly related concepts. In these cases, DSL specifications *include* domain concepts from other DSL specifications. To bridge the gap between the different domains, model-to-model transformations are used.

The models in CARM that are subject to co-evolution are mostly input and output models of model-to-model transformation unit tests. Whereas the models used during the development process, and on the lithography machine itself, are "reconstructed" from several artifacts in the software archive. The number of unit-test models are in an order of magnitude of hundreds, whereas the development models are in an order of magnitude of thousands. In the future, when the development models are no longer reconstructed, but become the primary source of specification, the co-evolution problem will increase in size.

IV. Evolution of Industrial MDSE Ecosystems

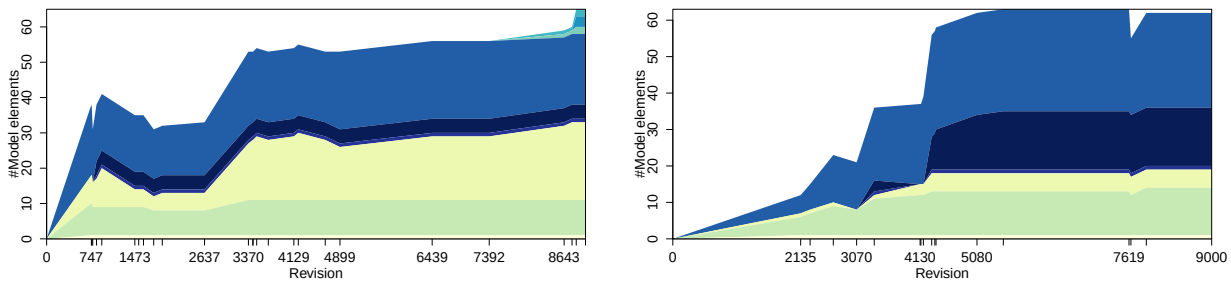
The DSLs in CARM evolve and become larger over time, as can be seen in Figure 4, in which the number of distinct modeling elements per language has been plotted over time. This evolution, gives rise to a large amount of co-evolutionary work with respect to the hundreds of model instances.

Furthermore, as a result of the language dependencies mentioned in Section III, co-evolution in the CARM ecosystem becomes more challenging than co-evolution in a collection of models in one DSL: changes made to a particular DSL might propagate to artifacts conforming to other DSLs. For example, the `Basics` DSL offers functionality for the declaration of connectable components using a variety of different communication ports and connections between them. These concepts are common throughout the application layer. Say we would



(a) Evolution of the `Basics` DSL, modeling functionality that common in the Application layer

(b) Evolution of the Directed Acyclic Graph (`DAG`) language, used for interaction with third party formal-analysis software



(c) Evolution the `ServoGroups` DSL, responsible for specifying groups of control blocks

(d) Evolution of the `AppMap` DSL, which specifies how application and platform are related

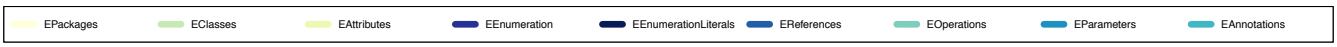


Fig. 4: Structures of ASML DSLs plotted over time; Different colors represent different types of meta-model elements.

remove a particular type of port (*e.g.*, a `dataPort`) from the language definition, then models from all languages in the application layer might be affected and require co-evolution. Additionally, transformations that transform the `dataPort` concept might need to be updated, as they contain redundant information (*i.e.*, how to transform a `dataPort`). This rippling effect of co-evolution as a consequence of language dependence can be observed in Figures 4a and 5. In Figure 4a, plotting the DSL structure of the `Basics` DSL over time, there is a large evolution around revision 4098. In Figure 5 the number of changed models of the `ControlBlocks` DSL has been plotted per revision. Here we see a large number of modifications around revision 4100 (visualized as light blue hexagons), relating to the evolution of `Basics`.

The need to co-evolve more models than those conforming to the meta-model being evolved makes meta-model evolution extra costly. Indeed, in some cases the induced co-evolution effort leads to meta-model evolution being postponed or to new domain concepts being encoded into legacy concepts. The latter practice jeopardizes the primary strength of DSLs: modeling in a specific domain. Advantages and promises of MDSE are, hence, being put at risk by the co-evolution costs.

This conclusion is further supported by the continuous change of the DSLs illustrated by Figure 4 akin to Lehman's

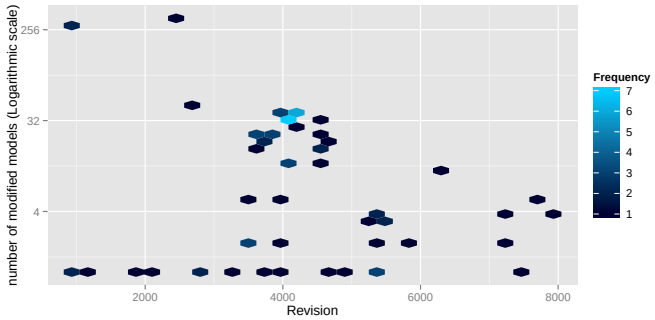


Fig. 5: Number of `ControlBlocks` models changed per revision.

law of continuing change [29].

To reduce the effort induced by meta-model evolution, a way-of-working similar to the butterfly method for database schema evolution [30] can be adopted. In that approach, a situation is created where both old and new concepts are supported, but modeling using legacy concepts is ceased. This way of working is illustrated in Figure 6. Using this approach, the immediate co-evolution pressure is decreased, as all models still conform to their meta-models. However, this

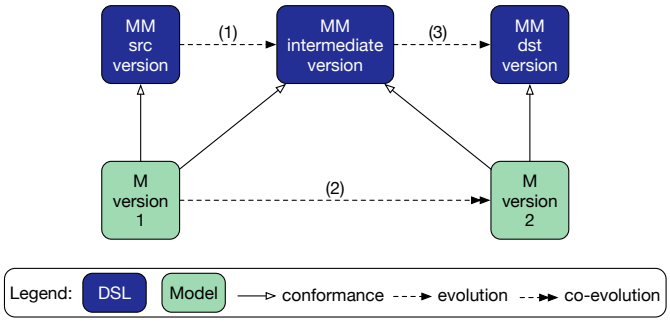


Fig. 6: To reduce the impact of DSL evolution, an intermediate version is created that supports both new and legacy concepts.

does not decrease the total amount of work that has to be done, and adds additional work for the creation of an intermediate DSL version (which is often tedious). Thus, the need arose for an approach with an increased level of automation.

V. Evaluation of the Operator-based Approach

Next we evaluate the operator-based approaches for application in industry. As far as we know, operator-based approaches have been evaluated in industry on at most four meta-models [4]. We investigate to what extent the operator based approach is usable for the evolution of large-scale industrial MDSE ecosystems (*i.e.*, twenty-two meta-models).

A. Operator Based Approach

In an *operator-based approach* [10] the user *specifies* the evolution of the meta-model using a set of pre-defined operators. Each of these operators can have a coupled operator if the meta-model evolution breaks conformance to the models. Such a coupled operator can be performed on model instances in order to mitigate the conformance-breaking effects of the evolution operator. For example, changing the type of an attribute in the meta-model from `EInt` to `EDouble` requires all values of this attribute in models to be changed to doubles. Edapt [31] (previously COPE [32]) is a tool that implements an operator-based approach.

Because evolution is primarily specified using a pre-defined set of operators, the practical applicability of an operator-based approach depends heavily on the available library of operators. The set of operators available (both in literature and in tooling) has been constructed in a demand-driven fashion: as the need for more operators arose through use cases [3], [4], they were constructed. Herrmannsdörfer *et al.* provide a library of 61 coupled operators [18], summarizing a number of these case studies. To the best of our knowledge, this is the most complete library available in literature.

To the extent of our knowledge, the operator-based approach has never been evaluated on an industrial case study of the size and complexity of CARM. An earlier study has shown that an operator-based approach is suitable for the evolution of up to four DSLs [3], [4]. We wish to investigate whether an operator-based approach is still feasible for specification of evolution on a large scale as indicated in Section III.

B. Experimental setup

To evaluate the usability of the operator-based approach in industry, we investigate which operators are required for the specification of the CARM use case and evaluate whether these operators are offered by the catalog of Herrmannsdörfer *et al.* [18], and the library of Edapt [31]. Although both these sources originate from the same research group, the catalog represents the academic view, whereas the library of Edapt is more practice oriented.

For now, we limit ourselves to *atomic* operators (also knows as primitives) [32], *i.e.*, operators with effects that cannot be decomposed into smaller operators on the meta-model. For example, the *compound* operator `CreateOppositeReference` [18] can be decomposed into an application of `CreateReference` and an application of `SetOppositeReference`, where the two latter operators cannot be decomposed further.

We restrict ourselves to atomic operators for two reasons. Firstly, an atomic operator encodes an atomic change on the meta-model. To be able to support every change on the meta-model, a complete set of atomic operators is required. Secondly, every compound operator can be expressed in terms of atomic operators. We aim to first make a complete library of atomic operators, and use this library to define compound operators later on.

To perform our evaluation, two distinct sets of data are required: the set of operators required for the the CARM use case, and the set of operators offered by the available libraries.

We first investigate which operators are required to specify the evolution of CARM. The tool EMFCompare [33] was used to compare subsequent pairs of revisions of the twenty-two DSLs stored in the ASML MDE repository. We have chosen to use EMFCompare for its ability to fine-tune comparisons in order to gain accuracy improvements [34]. Comparing all the subsequent revisions of all twenty-two DSLs yields a total of 3551 atomic changes represented in the EMFCompare difference model. In the remainder of the paper we refer to this set of changes as the *change history*.

Next, we wish to understand to what extent the available operator libraries cover the *change history*. We do so by automatically mapping every atomic operator offered by the libraries to a change in the EMFCompare difference model. This yields a set of changes in CARM that are covered by operators in literature, and a set of changes that are not.

C. Results

Of the twenty-two DSLs in CARM, only 19 in Table I have a history of changes. Languages in the Analysis, Application, Deployment, Mapping and Platform clusters are presented in Figure 3; for remaining languages in CARM having a history of changes we have made a Misc group.

1) Holistic analysis: From the analysis of our results, we observe that CARM requires support for 75 distinct atomic operations on meta-models in order to specify its evolution history. The catalog by Herrmannsdörfer *et al.* [18] supports 40 atomic operators, 32 of which are a subset of the 75 required

by CARM. These 32 operators together cover 85% of the *change history*. Edapt implements a slightly smaller subset of these operators, namely 28 operators, covering 81% of the *change history*.

An example of an atomic operator required for the specification of the *change history*, that is not supported by Herrmannsdörfer *et al.* and Edapt is the addition of an `EEnumLiteral`to an `EEnum`. Where operations for adding empty enumerations, moving enumeration literals between enumerations, and merging enumeration literals are supported, no operator exists for simply adding an enumeration literal. A more complex example is related to the `eKeys` of an `EReference`. An `EReference` may require all concepts referenced to be unique with respect to one or more attributes of those concepts, the so-called `eKeys` of a reference. The `eKeys` works similar to keys in the context of databases [35]. We see that `eKeys` are used throughout CARM DSLs (*e.g.*, to enforce each component in a machine to have a unique name). However, an operator for this is neither present in literature [18], nor in Edapt [31].

2) Differences between groups of DSLs: Having observed that at most 85% of our *change history* is supported by existing operator libraries, we also observe large difference between the coverage for different languages: for instance, the best covered language by the operators of Herrmannsdörfer *et al.* such as `PlatformMap`, `AppMap`, and `Deployment-Application` all exceed 95% of changes being covered, while for the worst covered languages `L3` and `Deployment-Mapping` the coverage values are 42% and 28%.

To obtain further insights in differences in coverage by the operator libraries we investigate whether any differences can be observed between different groups of DSLs. Indeed, DSLs have been grouped according to stacks and layers of the CARM ecosystem and can be seen therefore as representing the domain of the DSL. To this end we derive two contingency tables from Table I: the first contingency table (Table II) has language groups as rows, and changes covered (or not) by Herrmannsdörfer *et al.* [18] as columns. The second contingency table for changes covered (or not) by Edapt can be constructed in a similar way.

TABLE I: Coverage of languages in the CARM ecosystem

Group	DSL	#Changes	Covered by	
			[18]	[31]
Analysis	DAG	129	108	92
	Resource	39	34	34
	Schedule	226	206	203
Application	Application	183	158	154
	Basics	112	105	95
	ControlBlocks	1115	925	871
	ServoGroups	129	69	69
	TransducerGroups	88	77	77
Deployment	Deployment-Application	74	71	62
	Deployment-Mapping	25	7	7
	Deployment-Platform	107	105	99
Mapping	AppMap	178	170	141
Platform	LogicalPlatform	118	109	107
	PlatformMap	266	261	261
	PhysicalPlatform	507	482	462
Misc	L1	2	1	1
	L2	37	32	32
	L3	182	76	76
	L4	35	35	34

TABLE II: Contingency table for the operator coverage of DSL clusters by [18]. The numbers shown are the absolute amount of the *change history* covered

	Supported by [18]	Not supported by [18]
Analysis	348	45
Application	1334	293
Deployment	183	23
Mapping	170	8
Platform	852	39
Misc	144	112

Next we apply the χ^2-test of independence to each one of the contingency tables.[2] We use R, a free software environment for statistical computing, to perform statistical calculations [36].

The null hypotheses are therefore, H_0^H, the coverage of the operators by Herrmannsdörfer *et al.* [18] is independent of the DSL group; and H_0^E the coverage of the operators by Edapt is independent of the DSL group.

Both H_0^H and H_0^E can be rejected (the p-value was too small to be computed exactly). Hence, operator coverage, both for the library of Herrmannsdörfer *et al.* and of Edapt, is not independent of the DSL group. Closer inspection of the residuals reveals that for both operator libraries, coverage of the `Application` and `Misc` groups is lower than expected and `Platform` is higher than expected, whereas the `Platform` layer has a more traditional architecture, and is less complex. We conjecture that the lower coverage in the `Application` is related to the use of the *ontological instantation pattern* [37].

3) Operator libraries: We observed that overall 85% of the *change history* is covered by existing operator libraries, and important parts of the CARM ecosystem, such as the application layer, are being covered worse than expected. Hence, we conclude that the libraries of atomic operators currently available are insufficiently rich to specify the evolution of DSLs in a large-scale industrial MDE ecosystem. To mitigate the shortcomings in existing libraries, we propose to compute a complete set of atomic operators, rather than extend the available libraries in a demand-driven way.

VI. Completing the Operator-Based Approach

As stated before, we aim to derive a complete set of possible atomic operators for the evolution of Ecore-based metamodels. We do so by computing possible differences from the meta-meta-model and mapping these to atomic operators.

[2]Our choice for the χ^2 test is also the reason why we focus on groups of DSLs rather than on the individual DSLs. The χ^2 test requires each cell in the contingency table to exceed 5, and this would, for instance, not be the case for Interface/App that has merely three changes not covered by the operators of Herrmannsdörfer *et al.* [18].

A. Computation of Operators

First we determine all possible changes that can be made to a meta-model. For this, we use the fact that a meta-model is an instance of a meta-meta-model [16]. The number of features (i.e. attributes and references) in a meta-meta-model that can be instantiated to create an actual meta-model is limited. We call the features of the meta-meta-model that can be instantiated to create meta-models *instantiation points*. For example, the `name` attribute of an `EAttribute`, or the `isAbstract` attribute of an `EClass`.

Subsequently, we again use the EMFCompare difference model to encode possible changes on *instantiation points* of the meta-meta-model. This yields every possible way in which the instatiation of a meta-meta-model can be modified (*i.e.*, every way in which a meta-model can be altered). This set of changes to meta-meta-model *instantiation points* then has a one-to-one correspondence to the complete set of atomic operators. For example, an `ADD EClass` change on the *instantiation point* `EPackage.eClassifiers` corresponds to the "Add class to package" operator.

The algorithm for calculating the complete set of atomic operations is presented in Algorithm 1. The workings of this algorithm rely on the fact that Ecore meta-meta-model conforms to itself, and the core data structure of Ecore being a containment tree with cross references [26]. This allow us to use a method offered by the `EObject: eAllContents()`. The method `eAllContents()` returns the set of all `EObjects` recursively contained in a particular `EObject`. By calling `eAllContents()` on the Ecore root package, we can obtain all model elements that make up the Ecore meta-meta-model.

We then iterate over all structural features (*i.e.*, attributes and references), including inherited ones, of non-abstract classes in the Ecore meta-meta-model to find all possible *instantiation points*. More specifically, we wish to avoid structural features of abstract classes, as these do not occur in practice. We can easily iterate over all structural features of a class by using the `eAllStructuralFeatures` reference on every non-abstract class of the meta-meta-model.

When we have computed all *instantiation points*, we distinguish between a number of different operations that can be performed on the *instantiation points*, as prescribed by the EMFCompare difference model [33]:

1) If the feature consists of multiple elements (`isMany`), and the ordering of these elements is important (`isOrdered`), we support internal re-ordering of these elements via a `MOVE` operation.
2) If the feature is a containment reference, additive operations on these references actually introduce new elements into the model (as opposed to cross-references). We thus allow `ADD` and `DELETE` operations.
3) If the feature is a reference with an upper-bound of more than one, we are dealing with a collection. Values from collections can be added and deleted. We thus allow `ADD` and `DELETE` operations.

Data: *ecore* : The root package of the Ecore meta-meta-model
Result: $\mathcal{R}$: A set of atomic changes that can be performed on Ecore-based meta-models

```
R = ∅
foreach o ∈ecore.eAllContents() do
    if o instanceof EClass ∧ ¬o.isAbstract() then
        foreach f ∈ o.eAllStructuralFeatures() do
            if f.isChangeable() then
                t ← f.eType()
                if f.isMany() ∧ f.isOrdered() then
                    R ← R ∪ ⟨f, o, t, MOVE⟩
                end
                if f instanceof EReference then
                    if f.isContainment() ∨ f.upperBound =
                    -1 ∨ f.upperBound > 1 then
                        R ← R ∪ ⟨f, o, t, ADD⟩
                        R ← R ∪ ⟨f, o, t, DELETE⟩
                    else
                        R ← R ∪ ⟨f, o, t, CHANGE⟩
                    end
                else if f instanceof EAttribute then
                    R ← R ∪ ⟨f, o, t, CHANGE⟩
                end
            end
        end
    end
end
```

Algorithm 1: Algorithm for computing possible changes in a meta-model

4) If the feature is a reference, and the upper-bound is one, we only allow the `CHANGE` operation, using the same reasoning that is applicable to attributes.
5) If the feature is an attribute, its value can be changed (`CHANGE`). In the Ecore meta-meta-model, there are no occurrences of attributes with multiple values, (*i.e.*, `isMany` is always false), hence we need not support `ADD` and `DELETE`. Note that this might be the case for other meta-meta-models.

The algorithm generates a four-tuple for every atomic operator that can be performed on an Ecore-based meta-model. For example: a "rename class" change would be encoded as $\langle \texttt{EAttribute}(\textit{name}), \texttt{EClass}, \texttt{EString}, \texttt{CHANGE}\rangle$, and an "add attribute" change would be encoded as: $\langle \texttt{EReference}(\textit{eStructuralFeatures}), \texttt{EClass}, \texttt{EAttribute}, \texttt{ADD}\rangle$.

One thing to note is that the implementation of Ecore in Eclipse imposes additional constraints that are not captured in the meta-meta-model. For example, the `eType` of an attribute can theoretically have any `EClassifier`. In the graphical editor of Eclipse, it can only have an `EDataType` as its `eType`. However, the EMF API does allow for all the changes specified by our four-tuples to be performed, hence we consider $\mathcal{R}$ to be the set of all atomic operations that can

be performed on an Ecore-based meta-model.

Running the presented algorithm on the *Ecore* meta-meta-model yields a set of 213 atomic operators. The calculation of these operators is a one-time effort, only taking several seconds. In Section VI-B, we will compare this set of operators to the operators offered in literature [18] and by Edapt [31].

B. Discussion

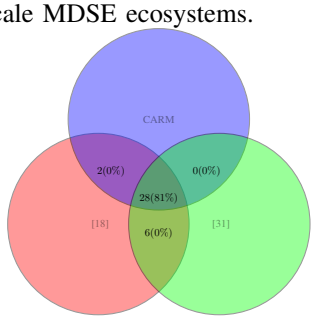
Using our methodology, we have generated 213 atomic operators, 75 of which are applicable to the CARM use case. A total of 3551 applications of these 75 operators are required for the complete specification of the *evolution history*. The comparison resulting from our study has been illustrated in Figure 7.

Of the 40 atomic operators supported by the catalog of Herrmannsdörfer *et al.* [18], 32 are applicable to the CARM use case. Together, they are able to specify 85% of the *change history*. Edapt [31] covers slightly less of the CARM use case. Edapt implements 32 of the operators by Herrmannsdörfer *et al.* that are useful for CARM, resulting in a specification coverage of 81% of the *change history*.

Additionally, we have identified 43 operators, that are neither available in the catalog of Herrmannsdörfer *et al.*, nor in the library offered by Edapt. Together these 43 operators cover 15% of the CARM *change history*. Among these 43 is the `eKeys` example mentioned in Section V-C.

Lastly, 138 operators are not used in specification of the CARM evolution. Of these 138, 127 are not available the library offered by Herrmannsdörfer *et al.* or in Edapt. We observe that among these 138 operators, 94 relate to annotations and operations in the meta-model. These modeling concepts are very scarce in the CARM use case, and the use cases in literature [3], [4].

Summarizing our results: Of the 213 operators theoretically possible, 75 are required for the specification of our *change history*, and only *32* are available. This leaves 43 operators left to be implemented before current operator libraries are sufficiently rich for specification of large-scale evolution specification. The remaining 138 operators, mainly concern `EOperations` and `EAnnotations`. We conjecture that these concepts are relevant for specification DSLs, and are thus of less importance for specification of evolution in large-scale MDSE ecosystems.



VII. Related work

In literature, a number of approaches have been proposed towards solving the co-evolution problem. These approaches

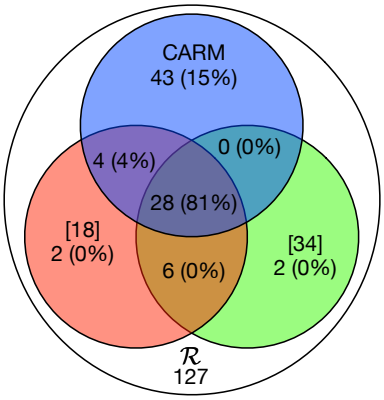


Fig. 7: Comparison of atomic operator support of our calculated set $\mathcal{R}$, a catalog of operators from literature [18], and Edapt [31]. Per subset we have presented the amount of operators present in this subset, and the percentage of the *change history* covered by these operators. Note that blue area corresponds to the *change history*.

TABLE III: Precise values per subset

(Sub)set	Num.	%
$\mathcal{R}$	213	100%
$\mathcal{R}$ \ CARM \ [18] \ [31]	127	0%
CARM	75	100%
[31]	36	81%
[18]	40	85%
CARM ∩ [18]	32	85%
CARM ∩ [31]	28	81%
CARM ∩ [18]∩ [31]	28	81%
CARM \ [18] \ [31]	43	15%
[18] \ [31] \CARM	2	0%
[31] \CARM \ [18]	2	0%
[18]∩ [31]	34	81%
[18]∩ [31] \CARM	6	0%

can be divided into a number of different categories:

Co-evolution oriented (*cf.* manual specification [17]) approaches consider the co-evolution specification to be the primary source of specification, and do not concern themselves with the evolution of meta-models. An example of such an approach is Epsilon Flock [38], which offers a DSL tailored towards co-evolution of models. Developers can use this DSL to specify co-evolution strategies for their models.

Evolution-oriented approaches aim to capture the essence of an evolution, and automatically derive a co-evolution specification from it. A number of different approaches to this have been described in literature:

1) **State-based approaches** [17] attempt to calculate the evolution between two versions of a meta-model supplied by the user (*e.g.*, `MMA` and `MMA'` in Figure 1). An example of such an approach is the EMFMigrate tool [39] developed at the university of L'Aquila.
2) **Operator-based approaches** [17] allow the user to specify the evolution of a meta-model using pre-defined operators. Each of these operators specifies part of a

TABLE IV: Fragment of comparison results

Operation	Value type	occurences in CARM	[18]	[31]
`CHANGE` reference type	`EClass`	571 (16%)	Yes	Yes
`CHANGE` attribute type	`EDataType`	367 (10%)	Yes	Yes
`ADD` reference to class	`EReference`	312 (9%)	Yes	Yes
`ADD` attribute to class	`EAttribute`	245 (7%)	Yes	Yes
`ADD` supertype	`EClass`	239 (7%)	Yes	Partial
`ADD` class to package	`EClass`	215 (6%)	Yes	Yes
`DELETE` reference from class	`EReference`	201 (6%)	Yes	Yes
`DELETE` attribute from class	`EAttribute`	141 (4%)	Yes	Yes
`DELETE` supertype	`EClass`	132 (4%)	Yes	Yes
`DELETE` class from package	`EClass`	119 (3%)	Yes	Yes
`ADD` a literal to an enumeration	`EEnumLiteral`	107 (3%)	Yes	No
`ADD` attribute to eKeys	`EAttribute`	32 (1%)	No	No
`CHANGE` namespace of a language	`EString`	18 (0.5%)	No	No
		76.5%		

meta-model evolution. Additionally, each of these operators can have a coupled operator that aims to mitigate conformance breaking effects the operator on the meta-model may have had. [7], [10]

3) **By-example approaches** have the user input a number of evolution examples. That is, for a number of models (*i.e.*, `alpha.MMA` in Figure 1) users present their evolved counterpart (*i.e.*, `alpha'.MMA'` in Figure 1). Subsequently they attempt to re-construct an evolution and co-evolution specification that meets the constraints imposed by the examples presented. [40]
4) **Generation approaches** aim at completely regenerating artifacts, rather than evolving them. An approach to this is using semantics from ontologies to migrate artifacts with respect to altered DSLs [41].

Among these approaches, a number of tools implement a variety of them. A feasibility study was preformed at ASML to evaluate applicability on the CARM use case. The tools under review are those reviewed by Herrmannsdörfer *et al.* [42]. Like Herrmannsdörfer *et al.*, we selected two tools as top candidates: Edapt (previously COPE) [31], [32], and Epsilon Flock [38]. These tools were selected based on maturity, stability, and application to the ASML use-case.

VIII. Conclusions & Future Work

In this paper, we have described a large industrial MDSE ecosystem, and identified (co-)evolutionary challenges arising from its size and complexity. We have found that co-evolution effort incurred by the evol utionary changes can put at risk the advantages and promises of the application of MDSE in industry.

To address this challenge, we have investigated to what extent an operator-based approach is feasible for solving the co-evolution problem. We have observed that the existing approaches [18], [31] cover up to 85% of the changes in the ecosystem history, and that 43 additional atomic operators need to be implemented to achieve 100% coverage.

Rather than extending an operator library in an *ad hoc* way, we have designed a top-down approach generating all possible atomic operators. On top of the existing operators and 43 operators that need to be implemented, the top-down approach revealed 127 additional operators. We conjecture that these operators express modification of meta-model elements that are less relevant for DSL specification. For example, in our ecosystem `EOperations` and `EAnnotations` elements have been used to improve performance of the source code generated from the models.

As future work, we aim to investigate more case-studies to establish to what extent this research generalizes. Furthermore, we will consider compound operators and extend the set of compound operators found in the literature [18] to support more co-evolution scenarios. Furthermore, we wish to extend the operator-based approach to support co-evolution of further artifact types, *e.g.*, model-to-model transformations. With respect to models, our goal is to support meta-model refactoring [12]. That is, when we only refactor the meta-model, *i.e.*, no expressivity is added or deleted, model co-evolution should be fully automatic.

Acknowledgment

This research is funded by ASML. The authors would like to thank the CARM development team for their cooperation in this research.

References

[1] "ASML." http://www.asml.com/. Accessed: 2015-04-07.
[2] G. Nordstrom, J. Sztipanovits, G. Karsai, and A. Ledeczi, "Metamodeling-rapid design and evolution of domain-specific modeling environments," in *Engineering of Computer-Based Systems. IEEE Conference and Workshop on*, pp. 68–74, 1999.
[3] M. Herrmannsdörfer, S. Benz, and E. Juergens, "Automatability of coupled evolution of metamodels and models in practice," in *Model Driven Engineering Languages and Systems* (K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, eds.), vol. 5301 of *Lecture Notes in Computer Science*, pp. 645–659, Springer Berlin Heidelberg, 2008.
[4] M. Herrmannsdörfer, D. Ratiu, and G. Wachsmuth, "Language evolution in practice: The history of GMF," in *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers* (M. van den Brand, D. Gasevic, and J. Gray, eds.), vol. 5969 of *Lecture Notes in Computer Science*, pp. 3–22, Springer, 2009.
[5] R. R. H. Schiffelers, W. Alberts, and J. Voeten, "Model-based specification, analysis and synthesis of servo controllers for lithoscanners," in *6th International Workshop on Multi-Paradigm Modeling*, (New York, NY, USA), pp. 55–60, ACM, 2012.
[6] T. Kühne, "Matters of (meta-) modeling," *Software & Systems Modeling*, vol. 5, no. 4, pp. 369–385, 2006.
[7] B. Gruschko, D. Kolovos, and R. Paige, "Towards synchronizing models with evolving metamodels," in *Proceedings of the International Workshop on Model-Driven Software Evolution*, 2007.

[8] A. Narayanan, T. Levendovszky, D. Balasubramanian, and G. Karsai, "Automatic domain model migration to manage metamodel evolution," in *Model Driven Engineering Languages and Systems* (A. Schürr and B. Selic, eds.), vol. 5795 of *Lecture Notes in Computer Science*, pp. 706–711, Springer Berlin Heidelberg, 2009.

[9] 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, 2008.

[10] G. Wachsmuth, "Metamodel adaptation and model co-adaptation," in *European Conference on Object-Oriented Programming 2007* (E. Ernst, ed.), vol. 4609 of *Lecture Notes in Computer Science*, pp. 600–624, Springer Berlin Heidelberg, 2007.

[11] J. García, O. Diaz, and M. Azanza, "Model transformation co-evolution: A semi-automatic approach," in *Software Language Engineering* (K. Czarnecki and G. Hedin, eds.), vol. 7745 of *Lecture Notes in Computer Science*, pp. 144–163, Springer Berlin Heidelberg, 2013.

[12] T. Levendovszky, D. Balasubramanian, A. Narayanan, and G. Karsai, "A novel approach to semi-automated evolution of dsml model transformation," in *Software Language Engineering* (M. van den Brand, D. Gaısević, and J. Gray, eds.), vol. 5969 of *Lecture Notes in Computer Science*, pp. 23–41, Springer Berlin Heidelberg, 2010.

[13] D. Di Ruscio, R. Lämmel, and A. Pierantonio, "Automated co-evolution of GMF editor models," *CoRR*, vol. abs/1006.5761, 2010.

[14] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Coupled evolution in model-driven engineering," *Software, IEEE*, vol. 29, pp. 78–84, Nov 2012.

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

[16] Z. Protić, *Configuration management for models: Generic methods for model comparison and model co-evolution*. PhD thesis, Eindhoven University of Technology, 2011.

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

[18] M. Herrmannsdörfer, S. D. Vermolen, and G. Wachsmuth, "An extensive catalog of operators for the coupled evolution of metamodels and models," in *Software Language Engineering* (B. Malloy, S. Staab, and M. van den Brand, eds.), vol. 6563 of *Lecture Notes in Computer Science*, pp. 163–182, Springer Berlin Heidelberg, 2011.

[19] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schema evolution: The PRISM workbench," *Proc. VLDB Endow.*, vol. 1, pp. 761–772, Aug. 2008.

[20] M. Lungu, "Towards reverse engineering software ecosystems," pp. 428–431, 2008.

[21] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems," in *Graph Transformations* (H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, eds.), vol. 7562 of *Lecture Notes in Computer Science*, pp. 20–37, Springer Berlin Heidelberg, 2012.

[22] B. Kienhuis, F. Deprettere, P. van der Wolf, and K. Vissers, "The y-chart approach," *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation-SAMOS*, vol. 2268, p. 18, 2002.

[23] P. van der Putten and J. Voeten, *Specification of reactive hardware/software systems*. PhD thesis, Eindhoven University of Technology, 1997.

[24] "SDF." http://www.es.ele.tue.nl/sdf3/. Accessed: 2015-04-07.

[25] "ESITrace." http://trace.esi.nl/. Accessed: 2015-04-07.

[26] "Ecore." http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details. Accessed: 2015-04-07.

[27] "Eclipse." http://www.eclipse.org/. Accessed: 2015-04-07.

[28] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2 ed., 2003.

[29] M. Lehman, "Programs, cities, students—limits to growth?," in *Programming Methodology* (D. Gries, ed.), Texts and Monographs in Computer Science, pp. 42–69, Springer New York, 1978.

[30] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan, "The butterfly methodology : A gateway-free approach for migrating legacy information systems," in *3rd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '97), 8-12 September 1997, Lake Como, Italy*, pp. 200–205, 1997.

[31] "Edapt." https://www.eclipse.org/edapt/. Accessed: 2015-04-07.

[32] M. Herrmannsdörfer, "Cope - a workbench for the coupled evolution of metamodels and models," in *Software Language Engineering* (B. Malloy, S. Staab, and M. van den Brand, eds.), vol. 6563 of *Lecture Notes in Computer Science*, pp. 286–295, Springer Berlin Heidelberg, 2011.

[33] "EMF Compare." https://www.eclipse.org/emf/compare/. Accessed: 2015-04-07.

[34] D. Kolovos, D. Di Ruscio, A. Pierantonio, and R. Paige, "Different models for model matching: An analysis of approaches to support model differencing," in *Comparison and Versioning of Software Models. ICSE Workshop on*, pp. 1–6, 2009.

[35] A. Silberschatz, H. Korth, and S. Sudarshan, *Database Systems Concepts*. New York, NY, USA: McGraw-Hill, Inc., 5 ed., 2006.

[36] R. Ihaka and R. Gentleman, "R: a language for data analysis and graphics," *Journal of computational and graphical statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[37] A. Laarman and I. Kurtev, "Ontological metamodeling with explicit instantiation," in *Software Language Engineering*, pp. 174–183, Springer, 2010.

[38] L. Rose, D. Kolovos, R. Paige, and F. Polack, "Model migration with epsilon flock," in *Theory and Practice of Model Transformations* (L. Tratt and M. Gogolla, eds.), vol. 6142 of *Lecture Notes in Computer Science*, pp. 184–198, Springer Berlin Heidelberg, 2010.

[39] J. Di Rocco, L. Iovino, and A. Pierantonio, "Bridging state-based differencing and co-evolution," in *Proceedings of the 6th International Workshop on Models and Evolution*, (New York, NY, USA), pp. 15–20, ACM, 2012.

[40] 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* (A. Düsterhöft, M. Klettke, and K.-D. Schewe, eds.), vol. 7260 of *Lecture Notes in Computer Science*, pp. 197–215, Springer Berlin Heidelberg, 2012.

[41] S. Roser and B. Bauer, "Automatic generation and evolution of model transformations using ontology engineering space," in *Journal on Data Semantics XI* (S. Spaccapietra, J. Pan, P. Thiran, T. Halpin, S. Staab, V. Svatek, P. Shvaiko, and J. Roddick, eds.), vol. 5383 of *LNCS*, pp. 32–64, Springer, 2008.

[42] M. Herrmannsdörfer and G. Wachsmuth, "Coupled evolution of software metamodels and models," in *Evolving Software Systems* (T. Mens, A. Serebrenik, and A. Cleve, eds.), pp. 33–63, Springer Berlin Heidelberg, 2014.

Science Reports **Department of Mathematics and Computer Science**
Technische Universiteit Eindhoven

If you want to receive reports, send an email to: wsinsan@tue.nl (we cannot guarantee the availability of the requested reports).

In this series appeared (from 2012):

12/01	S. Cranen	Model checking the FlexRay startup phase
12/02	U. Khadim and P.J.L. Cuijpers	Appendix C / G of the paper: Repairing Time-Determinism in the Process Algebra for Hybrid Systems ACP
12/03	M.M.H.P. van den Heuvel, P.J.L. Cuijpers, J.J. Lukkien and N.W. Fisher	Revised budget allocations for fixed-priority-scheduled periodic resources
12/04	Ammar Osaiweran, Tom Fransen, Jan Friso Groote and Bart van Rijnsoever	Experience Report on Designing and Developing Control Components using Formal Methods
12/05	Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse	A cure for stuttering parity games
12/06	A.P. van der Meer	CIF MSOS type system
12/07	Dirk Fahland and Robert Prüfer	Data and Abstraction for Scenario-Based Modeling with Petri Nets
12/08	Luc Engelen and Anton Wijs	Checking Property Preservation of Refining Transformations for Model-Driven Development
12/09	M.M.H.P. van den Heuvel, M. Behnam, R.J. Bril, J.J. Lukkien and T. Nolte	Opaque analysis for resource-sharing components in hierarchical real-time systems - extended version –
12/10	Milosh Stolikj, Pieter J. L. Cuijpers and Johan J. Lukkien	Efficient reprogramming of sensor networks using incremental updates and data compression
12/11	John Businge, Alexander Serebrenik and Mark van den Brand	Survival of Eclipse Third-party Plug-ins
12/12	Jeroen J.A. Keiren and Martijn D. Klabbers	Modelling and verifying IEEE Std 11073-20601 session setup using mCRL2
12/13	Ammar Osaiweran, Jan Friso Groote, Mathijs Schuts, Jozef Hooman and Bart van Rijnsoever	Evaluating the Effect of Formal Techniques in Industry
12/14	Ammar Osaiweran, Mathijs Schuts, and Jozef Hooman	Incorporating Formal Techniques into Industrial Practice
13/01	S. Cranen, M.W. Gazda, J.W. Wesselink and T.A.C. Willemse	Abstraction in Parameterised Boolean Equation Systems
13/02	Neda Noroozi, Mohammad Reza Mousavi and Tim A.C. Willemse	Decomposability in Formal Conformance Testing
13/03	D. Bera, K.M. van Hee and N. Sidorova	Discrete Timed Petri nets
13/04	A. Kota Gopalakrishna, T. Ozcelebi, A. Liotta and J.J. Lukkien	Relevance as a Metric for Evaluating Machine Learning Algorithms
13/05	T. Ozcelebi, A. Weffers-Albu and J.J. Lukkien	Proceedings of the 2012 Workshop on Ambient Intelligence Infrastructures (WAmIi)
13/06	Lotfi ben Othmane, Pelin Angin, Harold Weffers and Bharat Bhargava	Extending the Agile Development Process to Develop Acceptably Secure Software
13/07	R.H. Mak	Resource-aware Life Cycle Models for Service-oriented Applications managed by a Component Framework
13/08	Mark van den Brand and Jan Friso Groote	Software Engineering: Redundancy is Key
13/09	P.J.L. Cuijpers	Prefix Orders as a General Model of Dynamics

14/01	Jan Friso Groote, Remco van der Hofstad and Matthias Raffelsieper	On the Random Structure of Behavioural Transition Systems
14/02	Maurice H. ter Beek and Erik P. de Vink	Using mCRL2 for the analysis of software product lines
14/03	Frank Peeters, Ion Barosan, Tao Yue and Alexander Serebrenik	A Modeling Environment Supporting the Co-evolution of User Requirements and Design
14/04	Jan Friso Groote and Hans Zantema	A probabilistic analysis of the Game of the Goose
14/05	Hrishikesh Salunkhe, Orlando Moreira and Kees van Berkel	Buffer Allocation for Real-Time Streaming on a Multi-Processor without Back-Pressure
14/06	D. Bera, K.M. van Hee and H. Nijmeijer	Relationship between Simulink and Petri nets
14/07	Reinder J. Bril and Jinkyu Lee	CRTS 2014 - Proceedings of the 7th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems
14/08	Fatih Turkmen, Jerry den Hartog, Silvio Ranise and Nicola Zannone	Analysis of XACML Policies with SMT
14/09	Ana-Maria Şutîi, Tom Verhoeff and M.G.J. van den Brand	Ontologies in domain specific languages – A systematic literature review
14/10	M. Stolikj, T.M.M. Meyfroyt, P.J.L. Cuijpers and J.J. Lukkien	Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks
15/01	Önder Babur, Tom Verhoeff and Mark van den Brand	Multiphysics and Multiscale Software Frameworks: An Annotated Bibliography
15/02	Various	Proceedings of the First International Workshop on Investigating Dataflow In Embedded computing Architectures (IDEA 2015)
15/03	Hrishikesh Salunkhe, Alok Lele, Orlando Moreira and Kees van Berkel	Buffer Allocation for Realtime Streaming Applications Running on a Multi-processor without Back-pressure
15/04	J.G.M. Mengerink, R.R.H. Schiffelers, A. Serebrenik, M.G.J. van den Brand	Evolution Specification Evaluation in Industrial MDSE Ecosystems