Maintenance of Specification Models in Industry Using Edapt

Y. Vissers*, J.G.M. Mengerink*, R.R.H. Schiffelers*†, A.Serebrenik*, M.A.Reniers*,
*Eindhoven University of Technology, The Netherlands
†ASML, The Netherlands
Email: { j.g.m.mengerink, r.r.h.schiffelers, a.serebrenik, m.a.reniers }@tue.nl

***Abstract*—Domain specific languages (DSLs) ease the adoption of formal specification in industry. They allow developers to describe their specification models in concepts of their domain.**

However, DSLs evolve over time, causing specification models to have to co-evolve to reflect the evolution in the DSL. The maintenance overhead introduced by these, often manual, changes to specification models threatens to overshadow the advantages of DSL usage in industry. To this extent, many approaches have been proposed in the literature to facilitate DSL maintenance by automating model co-changes.

In this paper, we evaluate the ability of a tool, Edapt, to support the change and co-change in twenty-two industrial DSLs and corresponding specification models over a maintenance period of four years. We observe that the tool is only able to automatically co-change specification models for 72% of the DSL changes. To address the remaining 28% of the changes, we extend Edapt. The resulting extension allows automatically co-changing specification models for 98% of the DSL changes.

I. Introduction

A promising approach to designing formal specification languages is Model Driven Engineering (MDE). MDE allows developers to create Domain Specific Languages (DSLs) which enables modeling in terms of familiar domain concepts, rather than in generic concepts such as those offered by UML. The standardization of these MDE DSL formalisms [**?**] allows for fast creation and re-use of tools such as graphical editors and code generators. Due to these advantages, DSLs and MDE are increasingly being used for the specification of systems in industry [1], [**?**]. However, as the number of models created in a particular language grows, so do the maintenance challenges. Anecdotal evidence suggests that these maintenance challenges are threatening to become so large that they may overshadow the advantages of DSLs and MDE.

We focus on model maintenance with respect to DSL evolution. That is, DSLs may evolve over time [2], for instance due to new requirements or technical advancements of the domain. When DSLs evolve, models created in those DSLs might need to co-evolve to reflect the evolution of the language, as illustrated in Fig. 1. This challenge is similar to database maintenance with respect to a schema evolution [3], or code refactoring with an API evolution [**?**]. As the number of models for a particular DSL may grow into the hundreds (as illustrated in Fig. 2), manual maintenance of models becomes infeasible, calling for an automated solution.

In the literature, many approaches and tools have been proposed towards automating model maintenance. As our

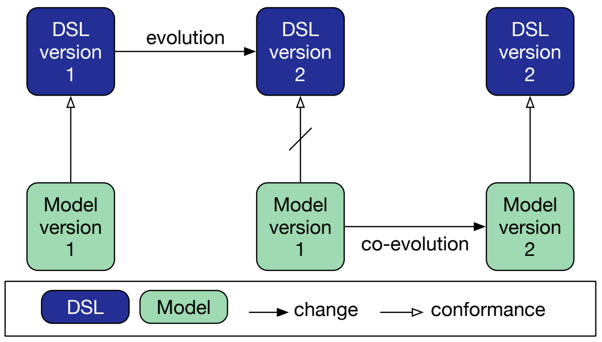

Fig. 1: A schematic overview of the DSL maintenance problem, also known as the co-evolution problem

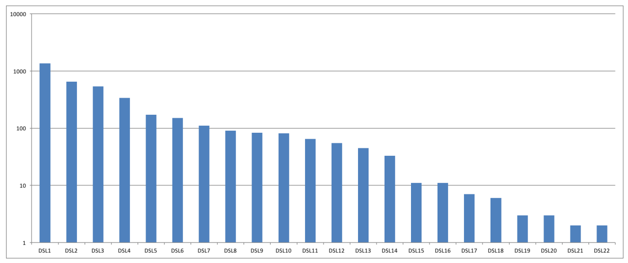

Fig. 2: The number of models per DSL *on a logarithmic scale.*

research takes places in industry, we require a mature tool with a low learning curve that enables reuse. To meet these requirements, we have selected Edapt[1], which is the official Eclipse tool for migrating EMF models in response to DSL evolution [5].

Edapt is a tool that implements the operator-based approach. The operator-based approach captures frequently occurring patterns of DSL evolution and model co-evolution in terms of reusable *operators*. The usability of this approach in large scale applications thus depends heavily on the amount of reusable operators available. Earlier work [6] has shown that the state-of-the-art operator library [7] lacks 11% of the operators required to automatically (co-)evolve the DSLs and their models. We suspect that Edapt has similar deficiencies. Thus we pose our first research question:

[1]Previously COPE [4]

RQ1: To what extent is Edapt able to automatically (co-)evolve DSLs and models in an industrial context?

We conclude that Edapt is only able to specify 72% of the (co-)evolution in the sample DSLs chosen using reusable operators. Having identified the deficiencies in Edapt, we wish to utilize the extensibility of Edapt and implement additional operators to mitigate the deficiencies identified. Thus, we pose our second research question:

RQ2: What challenges arise when extending the library of reusable operators provided by Edapt?

We classify the operator deficiencies (*i.e.,* operators required by our case, but not offered by the library) of Edapt with respect to their degree of automatability [8]. The least automatable operator deficiencies require user interaction. We follow Herrmannsdörfer [9] and implement user interaction as a choice between a number of provided options. Having extended Edapt with a number of operators that require user interaction, Edapt is now able to automatically perform 98% of the DSL/model (co-)evolutions. **We conclude that, once extended with additional operators, and support for model-specific operators, Edapt is suitable for the maintenance of DSLs and models in an industrial context.**

The remainder of this paper is structured as follows. In Section II we discuss the industrial background of our research as well as operator-based co-evolution. In Section III we elaborate on the experimental choices made for answering our research questions, the results of which we present in Section IV. Lastly, we present our conclusions in Section V.

II. Background

A. Introduction to DSLs and their Evolution

Before going into detail about DSL evolution, in this section we present an example of a DSL, its evolution, and the model co-evolution challenge it poses.

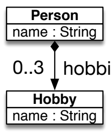

Fig. 3: A sample DSL before evolution

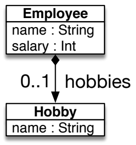

Fig. 4: A sample DSL after evolution

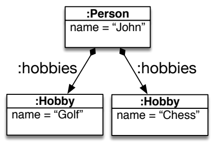

Fig. 5: A sample model for original DSL

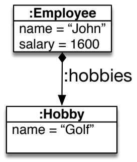

Fig. 6: The evolved version of the model , that is now valid for the evolved DSL

To start, an original version of the sample DSL can be defined by the class diagram illustrated in Figure 3. This DSL allows for the specification of a person with hobbies. Note that a person has up to three hobbies. An example model for this DSL is illustrated in Figure 5.

We can now evolve this DSL, as illustrated in Figure 4, to represent employees. This evolution consists of three steps: (1) the concept `Person` is renamed to `Employee`, (2) `Employee` gets a `salary` attribute, and (3) the maximum number of `hobbies` is reduced to one.

The example `Person` model in Figure 5 is now no longer valid for a number of reasons. Firstly, the concept `Person` is unknown in the context of the DSL in Figure 3. Secondly, a `Person` can now only have one hobby, and the model has two. Lastly, there is no salary attribute in the primary model concept.

To co-evolve the model in response to the DSL evolution, the instance of `Person` must be retyped to be a person of `Employee`, the instance of `Employee` must have its salary attribute set, and one of its hobbies must be removed. The result of this co-evolution is illustrated in Figure 6.

In this small example, the number of steps required to co-evolve the model are still limited. However, if several hundred models require similar (but often not identical) co-evolution, this task quickly becomes tedious, error-prone, and thus costly.

B. Industrial Context

Our research takes place at ASML, provider of lithography equipment for the semiconductor industry. In recent years, MDE is gradually being incorporated into the development process at ASML. In particular, we look at the CARM ecosystem [1].

The CARM ecosystem consists of twenty-two DSLs to model servo control components at various levels of abstractions, using the Y-chart paradigm for system decomposition [10]. The DSLs in CARM are defined as EMF Ecore class diagrams [2] [11], [12] and OCL constraints [13]. Both the EMF Ecore class diagrams and OCL constraints have evolved actively over the past four years. Fig. 7 shows the evolution of an EMF Ecore class diagram for a CARM DSL.

The models that are currently affected most frequently by the DSL evolution are models that reside with the *DSL developer* (cf. Figure 2), which have to co-evolve with every evolution of the DSL during its development. Moreover, thousands of models residing with the *model developers* should co-evolve whenever a new DSL version is released. Manual co-evolution of these models is a tedious, error-prone, and therefore costly, process [14], [6]. Due to the DSL developers and model developers having several, separate, sets of models, the maintenance effort is shared between several parties and additional communication and synchronization overhead has to be incurred during maintenance.

[2]Also known as meta-models

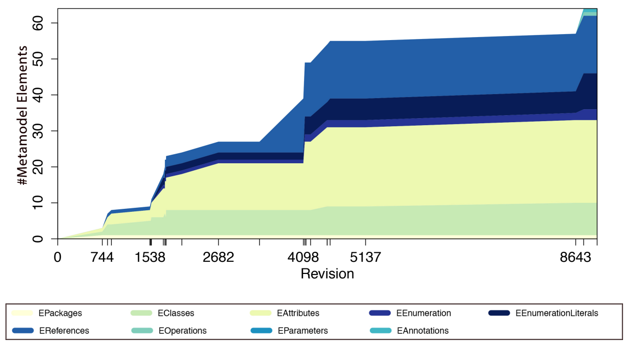

Fig. 7: The size and structure of a DSL in the CARM ecosystem over time (where time is represented by revisions from the repository)

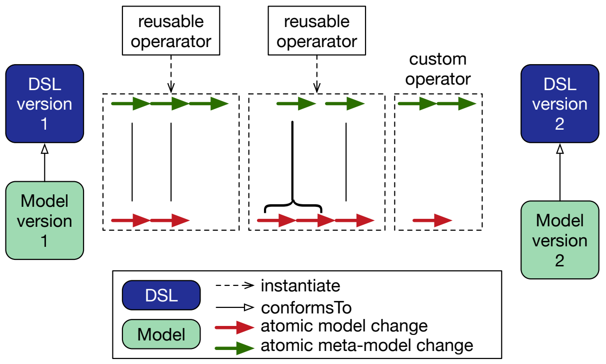

Fig. 8: A schematic overview of coupled evolution, using a sequence of re-usable and custom operators

C. Edapt: Operator-Based Co-Evolution

Edapt [15] (previously developed as COPE [14], [16]) is a tool for model co-evolution that implements the operator-based approach, and is integrated into the Eclipse Modeling Framework (EMF) [11].

Operator-based model co-evolution is performed incrementally by executing a sequence of coupled operators that encode one or more DSL evolutions in addition to zero or more model co-evolution steps [15]. The DSL developer specifies a sequence of operators from two types (Fig. 8):

- Re-usable operators are operators that can be quickly configured to perform frequently occurring patterns of coupled evolution (*e.g.,* rename a class).
- Custom operators allow users to manually specify a pattern of coupled evolution when no reusable operator is available, or to add custom semantics to existing operators (*e.g.,* rename an integer attribute and square all its values).

The usability of the operator-based approach relies on the availability of re-usable operators [17]. When no reusable operator is available, a custom operator has to be manually created to mitigate this deficiency, increasing the required co-evolutionary effort.

D. Related work

This study is based on our previous work [**?**]. While in that work we have focused on constructing and evaluating a complete operator library $\mathcal{R}$ for DSL evolution specification, the current study performs a similar evaluation of Edapt.

III. Study Setup

To answer **RQ1**, we perform a *conceptual replication* (*cf.* [18]) of the work of Herrmannsdörfer *et al.* [8]. We perform our case study by investigating to what extent the evolution of the twenty-two CARM DSLs can be specified using the reusable operators from Edapt.

To do so, we require a specification of the evolution of the DSLs in the CARM ecosystem. We use EMFCompare [19] to determine differences in subsequent versions pairs of the DSLs, obtained from the subversion (SVN) repository of ASML. This yields a collection of pairwise differences (= changes) between all subsequent DSL versions in the history of the ecosystem, which we refer to as the *evolution history*.

A known threat to validity of construction of the evolution history is the accuracy of EMFCompare [8]. To mitigate this threat, we have manually verified a subset of the EMFCompare results [20]. The results of EMFCompare were found to be perfect, with the exception of detecting renames.

To answer **RQ2**, we classify the operator deficiencies of Edapt with respect to CARM, using the classification presented by Herrmannsdörfer *et al.* [8], presented in Fig. 10. We observe that most challenges are related to the lowest automatability class: *model-specific* [8]. To overcome these challenges, we extend Edapt following the approach of Herrmannsdörfer *et al.* [9].

IV. Results

In this section, we present and discuss our findings with regard to the previously posed research questions.

A. Reusable Operators

For **RQ1** we study to what extent the reusable operators of Edapt cover the evolution of the CARM case study. More specifically, we investigate what operators are required to specify the evolution of the CARM ecosystem during the four year development history, and asses whether Edapt supports these operators.

The *evolution history* consists of 3405 DSL changes, such as “Rename Class Person to Employee”, that can be described using 70 operators, such as “Rename Class”. We observe that only 28 (40%) of these 70 operators are supported by Edapt. However, plotting the number of DSL changes per operator in the evolution history, as shown in Fig. 9, reveals that Edapt supports most of the frequently used operators. Closer inspection reveals that of the 3405 DSL changes in the evolution history, 827 (24%) are not supported at all by Edapt, and 138 changes (4%) are only partially supported. For instance, one of the 70 operators required by the CARM

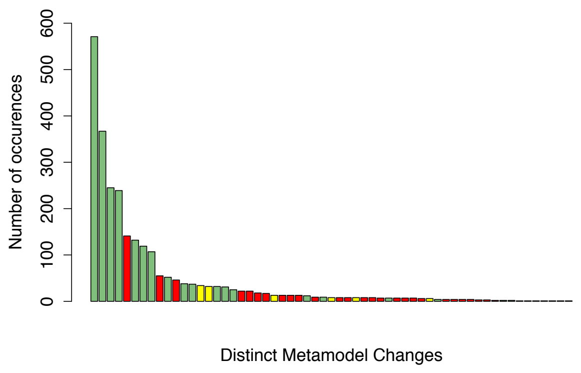

Fig. 9: Number of applications per operator required to specify the *evolution history*, where operators supported, partially supported, and not supported are colored green, yellow, and red respectively.

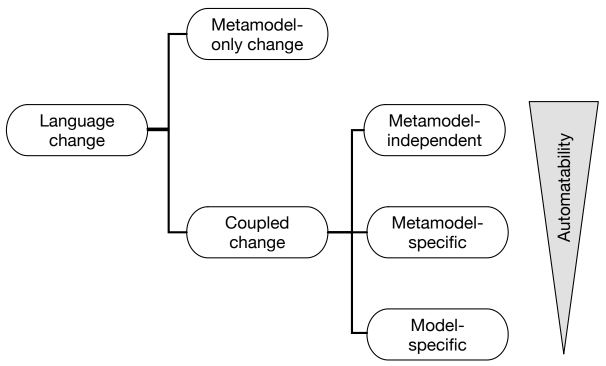

Fig. 10: A classification of DSL evolutions with respect to their model co-evolution automatability [8]: meta-model-independent (MMI), meta-model-only (MMO), meta-model-specific (MMS) and model-specific (MS)

evolution history is "change upperbound of an attribute". However, Edapt only supports increasing the upper bound of an attribute, rather than changing it in general. Hence, we consider the operator "change upper bound" to only be partially supported by Edapt.

The 2440 changes that are supported by an operator in the library of Edapt constitute 72% of the evolution history, contrasting the earlier claim that Edapt can cover over 90% of DSL changes [15].

B. Qualification of DSL changes

Forty-two of the operators used in the evolution history are not supported by Edapt. For example, creating a new literal in an enumeration, or specializing the bounds of an attribute (increasing the lowerbound or decreasing the upperbound). To identify how challenging automation of these operators will be, we classify them with respect to their automatability, using the classification of Herrmannsdörfer in Fig. 10 [8], and answer **RQ2**.

Herrmannsdörfer has identified the following automatability classes:

- Meta-model-only[3] operators do not require a model co-evolution. These operators perform a conservative extensions of a DSL with respect to its models.
- Coupled operators do require models to co-evolve to reflect changes. Herrmannsdörfer distinguishes between
  - Meta-model-independent[4] operators have a characteristic co-evolution specification, which can be applied to any DSL. Additionally, the co-evolution specification can be used to co-evolve any model. As a result, meta-model-independent changes have a very high potential for automation.
  - Meta-model specific[5] operators require a co-evolution specifiation that is specific to the meta-model. Therefore the co-evolution specification cannot be re-used, but can be used to co-evolve all models for the particular meta-model.
  - Model-specific operators form the biggest challenge in co-evolution. The co-evolution specification varies between models, because model-specific information injection is required to obtain the co-evolution for each model. Note that it is possible to have model-specific operators that can be applied to multiple DSLs.

Of the operators identified under **RQ1**, we disregard operators related to reordering, and EOperations, as these do not influence model conformance. Furthermore, operators pertaining to EAnnotations were not implemented, as they are not essential in defining the structure of DSLs. Of the remaining operators, we implement all operators that have more than one occurrence in the evolution history. These, top fourteen, operators are presented in Table I, and include seven meta-model-only (MMO), one meta-model-independent (MI), and six model-specific (MS) operators.

The qualification of the operators was performed manually and is thus susceptible to error. To validate its quality, the qualification was performed by the first and the second authors independently. Both qualifications were compared and no discrepancies were found.

We observe that the majority of the 72% not supported by Edapt and 4% partially supported by Edapt (Section IV-A) are reusable model-specific operators. This contrasts the claim that Edapt supports $> 90\%$ of DSL evolutions with reusable operators, and leads us to extend Edapt with additional operators.

C. Model-Specific Operators with User Interaction

As mentioned in Section IV-A, we identified six model-specific operators required by the CARM evolution history, that are absent from Edapt. The main characteristic of these operators is that each model (may) require a different co-evolution. Therefore, an information injection is required to

[3]DSL-only: a meta-model described a DSL
[4]DSL-independent: a meta-model described a DSL
[5]DSL-specific: a meta-model described a DSL

TABLE I: A summary of operators missing from the library of Edapt that have been implemented by us.

Operator Name	Element	Parameters	Class
Create EEnumLiteral	EEnum	name:EString, literal:EString, value:EInt	MMO
Change EENum-LiteralValue	EEnumLiteral	value:EInt	MMO
Set Opposite Reference	EReference	eOpposite:EReference	MMO
Change Default Value Literal	EAttribute	defaultValueLiteral :EString	MMO
Drop Attribute ID	EAttribute	id:EBoolean	MMO
Make non-abstract	EClass	isAbstract:EBoolean	MMO
Change interface	EClass	isInterface:EBoolean	MMO
Change NS Prefix	EPackage	nsPrefix:EString	MMI
Specialize Attribute bounds	EAttribute	lowerBound:EInt, upperBound:EInt	MS
Set Supertype	EClass	supertype:EClass	MS
Set Keys	EReference	eKeys:List, EAttribute	MS
Set Attribute ID	EGenericType	eClassifier:eClassifer	MS
Add TypeArgument	EGenericType	eTypeArgument : EGenericType	MS

indicate how to co-evolve each model. To allow for reusability of the model-specific operators, they have to be specified in a generic way, such that they can be configured for any DSL and can be used to co-evolve any model of that DSL. This is achieved with configurable parameters to select the relevant DSL concepts.

For example, consider an operator that specializes the bounds of a (multi-valued) attribute, *e.g.,* a class `Person` which has any number of `hobbies` (attribute). Using an operator, the class `Person` is now restricted to having either one or two hobbies. When this operator is applied, the models that have an instance of `Person` with too few or too many hobbies will no longer conform to the evolved DSL. As the operator must be able to co-evolve every model of the DSL, but there is no way for the operator to know beforehand which values should be removed or added, additional information is required (*i.e.,* information injection). We consider this to be the primary challenge in extending the library of Edapt (**RQ2**).

To overcome this challenge, we adopt the approach of Herrmannsdörfer *et al.* based on user interaction during co-evolution as described in [9]: "the migration algorithm automatically migrates the model as far as possible, and whenever it needs supplementary information, it asks the language user to provide the missing information". The user should have the possibility to either inject the information for each model individually, or give an expression that refines the co-evolution specification such that implicit information (*e.g.,*, naming conventions) can be used to co-evolve all models. In [9], Herrmannsdörfer *et al.* demonstrates a user interface for interactive coupled operators in COPE. Because this interface is not part of Edapt, we prototyped a similar concept. In this prototype, if conformance is broken during co-evolution, the user is provided with a dialog to provide the relevant information for restoring conformance. These extensions to Edapt allow us to automatically co-evolve our models in response to 98% of the DSL evolutions in our four year history.

V. Conclusions

In this paper, we have evaluated the extent to which Edapt can perform evolution and co-evolution of DSLs and models in a large-scale industrial ecosystem of DSLs. We observe that, in contrast to the earlier claims of Edapt with respect to supported evolutions ($> 90\%$), only 72% of DSL evolutions are fully supported. Another 4% is partially supported (**RQ1**). This discrepancy is primarily caused by a large demand for model-specific operators in our case study, while such a demand is absent from earlier case studies by Herrmannsdörfer *et al.* [15], [8]. We conjecture that this increase in demand for model-specific operators is caused by the larger number of languages, and the longer maintenance history of our case study. These larger numbers may account for an increase in less frequent evolutions, requiring model-specific operators.

To cope with the operator deficiencies in Edapt, we have implemented the most frequently occurring model-specific coupled operators using the methodology described by Herrmannsdörfer *et al.* [9] (RQ2).

After the addition of these (model-specific) coupled operators, Edapt was able to specify 98% of the coupled evolution in the CARM case study. The remaining two percent consists of a large number of operators that have a limited number of occurrences over the course of four years. Thus, we conclude that the model co-evolution features provided by Edapt make it suitable for the maintenance of DSLs and models in industry. Provided that Edapt is extended with additional reusable operators and support for model-specific operators (for instance by means of user interaction).

Acknowledgment

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

References

[1] 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.

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

[3] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schema evolution: The PRISM workbench," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 761–772, Aug. 2008.

[4] M. Herrmannsdörfer, in *SLE*, ser. LNCS, B. Malloy, S. Staab, and M. G. J. van den Brand, Eds. Springer, 2011, vol. 6563, pp. 286–295.

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

[6] J.G.M. Mengerink, R.R.H. Schiffelers, A. Serebrenik, and M.G.J. van den Brand, "Evaluation of (co-)evolution specification in industrial mdse ecosystems," Eindhoven University of Technology, Tech. Rep. CSR-15-04, 2015.

[7] 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, B. Malloy, S. Staab, and M. G. J. van den Brand, Eds. Springer, 2011, vol. 6563, pp. 163–182.

[8] M. Herrmannsdörfer, S. Benz, and E. Juergens, "Automatability of coupled evolution of metamodels and models in practice," in *Model Driven Engineering Languages and Systems*, ser. LNCS, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds. Springer, 2008, vol. 5301, pp. 645–659.

[9] M. Herrmannsdörfer and D. Ratiu, "Limitations of automating model migration in response to metamodel adaptation," in *MSE, Workshops and Symposia at MODELS*, ser. LNCS, vol. 6002. Springer, 2009, pp. 205–219.

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

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

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

[13] J. Warmer and A. Kleppe, *The Object Constraint Language: Getting Your Models Ready for MDA*, 2nd ed. Addison-Wesley, 2003.

[14] M. Herrmannsdörfer, S. Benz, and E. Juergens, *ECOOP*. Springer, 2009, ch. COPE—Automating Coupled Evolution of Metamodels and Models, pp. 52–76.

[15] "Edapt," https://www.eclipse.org/edapt/, accessed: 2015-04-07.

[16] M. Herrmannsdoerfer, S. Benz, E. Juergens *et al.*, "COPE: A language for the coupled evolution of metamodels and models," in *International Workshop on Model Co-Evolution and Consistency Management*, 2008.

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

[18] 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.

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

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