DSL/Model Co-Evolution in Industrial EMF-Based MDSE Ecosystems

J.G.M. Mengerink
Eindhoven University of Technology
The Netherlands
j.g.m.mengerink@tue.nl

R.R.H. Schiffelers
ASML & Eindhoven Univerisity of Technology
The Netherlands
r.r.h.schiffelers@tue.nl
ramon.schiffelers@asml.com

A. Serebrenik
Eindhoven University of Technology
The Netherlands
a.serebrenik@tue.nl

M.G.J. van den Brand
Eindhoven University of Technology
The Netherlands
m.g.j.v.d.brand@tue.nl

ABSTRACT

Model Driven Engineering and Domain Specific Languages (DSLs) are being used in industry to increase productivity, and enable novel techniques like virtual prototyping. Using DSLs, engineers can model a systems in terms of their domain, rather than encoding it in general purpose concepts, like those offered by UML. However, DSLs evolve over time, often in a non-backwards-compatible way with respect to their models. When this happens, models need to be co-evolved to remain usable. Because the number of models in an industrial setting grows so large, manual co-evolution is becoming unfeasible calling for an automated approach. Many approaches for automated co-evolution of models with respect to their DSLs exist in literature, each operating in a highly specialized context. In this paper, we present a high-level architecture that tries to capture the general process needed for automated co-evolution of models in response to DSL evolution, and assess which challenges are still open.

Keywords

Model driven engineering, evolution, domain specific language, model, maintenance, co-evolution

1. INTRODUCTION

Model driven (system) engineering MD(S)E is being used both in industry [1, 2], and open source [3] for developing software and systems. Systems design using MDE allows for analysis and feedback early in the design process. A main driver for doing so is designing domain specific languages (DSLs), *e.g.,* using the Eclipse Modeling Framework (EMF) [4]. In EMF, DSLs consist of meta-models [5] eventually augmented by OCL constraints [6]. DSLs allow one to model systems in terms of their domain, rather then using general purpose concepts offered by, *e.g.,* UML or SysML [7].

We have observed that in an industrial setting, DSLs often occur in an ecosystem (*cf.* [8, 9, 10]), *i.e.,* collection of DSLs with dependencies between them introduced,*e.g.,* by language reuse or model-to-model transformations. Additionally, infrastructural artifacts such as parsers, textual editors, and graphical editors also reside in this ecosystem.

Similarly to traditional software systems and languages [11], DSLs evolve over time [12, 13]. This leads to challenges with respect to backwards compatibility: artifacts (models, parsers etc.) from the old version of the DSL may no longer be valid in the new version of the DSL. To this extent, these artifacts have to co-evolve to reflect the changes in the DSL. This is known as the *co-evolution problem*.

Manual co-evolution of these artifacts is tedious, error-prone, and costly. To mitigate this, we wish to automate the co-evolution of artifacts in DSL ecosystems to the highest extent possible. In the literature, various approaches have been presented towards automating this co-evolution, each with their own strong and weak points [14, 15, 16, 17]. For our industrial case we aim at completeness and formality, rather than approximation of the various artifacts.

In this paper, we present *a high-level architecture that captures the various steps and components required for co-evolution of models*. Subsequently, we examine the state-of-the-art in co-evolution research to ascertain to what extent the state-of-the-art is able to effectuate the proposed architecture. Lastly, we summarize the open challenges towards implementing the architecture, and are thus future work for automating DSL/model co-evolution.

In the remainder of this paper we discuss DSL ecosystems (Section 2),and present our architecture and its components (Section 3), position existing work with respect to the architecture and determine which components are not yet supported (Section 4). Next we discuss the theoretical limitations of the automation (Section 5) and sketch the directions for future work (Section 6).

2. ECOSYSTEMS

In model-driven engineering, the meta-model is the central artifact that dictates the concepts and structure of other artifacts. Several related DSLs and their corresponding artifacts constitute an MDSE ecosystem. When evolving a DSL in an MDSE ecosystem, artifacts such as models [14, 15, 18, 19], model-to-model transformations [20, 21], text and graphical editors [22] may have to be co-evolved. Di Ruscio, Iovino and Pierantonio define three categories of co-evolving

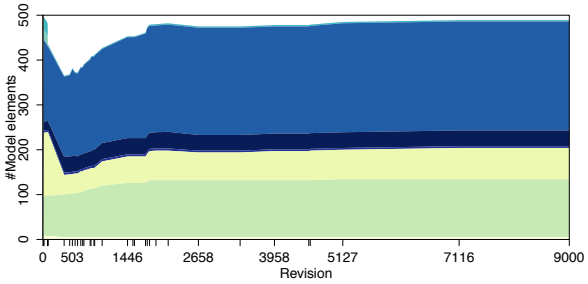

Figure 1: Evolution of the number of meta-model elements in `PGWB`. Largest groups represent references (blue), enumeration literals (indigo), attributes (yellow), classes (green). Numerous changes to the meta-model are visible.

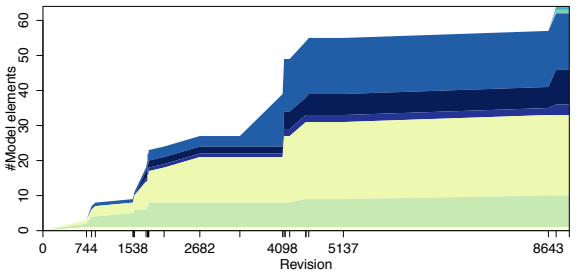

Figure 2: Evolution of the number of meta-model elements in `Basics`. Largest groups represent references (blue), enumeration literals (indigo), attributes (yellow), classes (green). Numerous changes to the meta-model are visible, in addition to a gradual increase in size.

artifacts [23]: DSL[1]/Model co-evolution, DSL/Transformation co-evolution, and DSL/Editor co-evolution.

The driving cases behind our research are several MDSE ecosystems at ASML, provider of lithography equipment for the semiconductor industry. The largest ecosystem we consider is CARM [2] consisting of 22 EMF-based DSLs, 95 QVT model transformations, and 5500 unit-test models supporting development of these transformations [24].

In the CARM ecosystem, models make up the majority of artifacts. Thus, in this work we focus on DSL/model co-evolution. The model co-evolution effort required by a DSL evolution can be expected to become bigger when carried out in the context of an MDSE ecosystem as opposed to a single DSL and its models. The reasons are threefold: reuse of concepts from other DSLs, model transformations and implicit relations between DSLs.

Indeed, meta-models may *reuse* concepts from other meta-models. However, let meta-model X reuse a concept A from meta-model Y. If A in Y evolves, models of meta-model X reusing A (from Y) might need to co-evolve. Hence, evolution of a single DSL, can cause co-evolution of models in other DSLs.

Similar ripple effects [25, 26] might be caused by *model transformations*. Assume a transformation T transforms a concept A in meta-model X to a concept B in Y. If A is evolved to include additional information, one might have to co-evolve T to incorporate this new information. Additionally, Y might not be expressive enough to encode the new information, and B itself has to be co-evolved too.

[1]We consider a DSL to consist of a meta-model enriched with OCL constraints. The original work of Di Ruscio, Iovino and Pierantonio considered meta-models only.

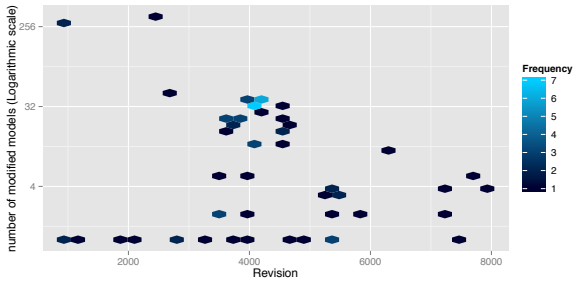

Figure 3: A binplot (aggregating several data-points into hexagonal bins) of the number of revised `PGWB` models per revision. Each hexagon represents a bin of a range of revisions and a range of "number of models modified", where the colors represent the number of data points in that bin.

Lastly, not all relations between artifacts in MDSE ecosystems are modeled (or specified) explicitly. There can also be *implicit relations*, such as the classic software co-change as described by Zimmerman *et al.* [27]. Similar relations exist for MDSE and similar solutions can be presented [28].

To illustrate the ripple effect issue in CARM consider DSLs `PGWB` and `Basics`. `PGWB` reuses concepts from `Basics`. Evolution of `PGWB` and `Basics` is shown in Figures 1 and 2, respectively. Around revision 4100 we see an increase in the number of classes, enumeration literals and references in `Basics` language. The corresponding change in `PGWB` is barely visible. However, Figure 3 shows that there is a large number of `PGWB` models that are being co-evolved around revision 4100, in response to the evolution in `Basics`.

This example illustrates that presence of inter-DSL dependencies in an MDSE ecosystem causes a ripple effect and increases costs of manual maintenance. Hence, an automatic approach is required to facilitate co-evolution of artifacts in MDSE ecosystems.

3. A HIGH-LEVEL ARCHITECTURE FOR CO-EVOLVING MODELS

As described in Section 2, several types of artifacts must co-evolve in response to DSL evolution. In this section we present a high-level architecture for co-evolving models in response to DSL evolution. However, a similar architecture may be used for other artifacts. We focus on models (rather than model transformations or editors) since models make up the majority of CARM artifacts. Moreover, we focus on co-evolving a single (arbitrary) model in response to the evolution of a single DSL. In the case where there are multiple DSLs, we can treat them as a single DSL by resolving all inclusions and dependencies. When more models are involved, the proposed process can simply be repeated for each model individually, as we have no assumptions on the model, other than that it is a valid model for the first version of the DSL.

Figure 4 presents a high-level architecture for model co-evolution. The process starts when a DSL (1) evolves (2) to

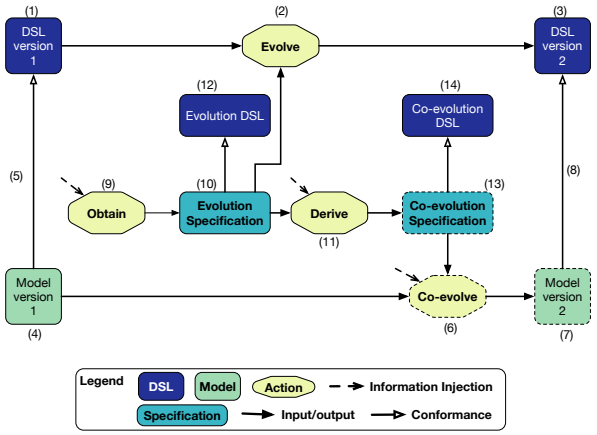

Figure 4: The framework for positioning research(questions) with respect to DSL/model co-evolution.

Table 1: A summary of the components of the high-level architecture that have to be implemented.

Comp.	Description	Fig. 4
C1	A way to model meta-model evolution	12
C2	A way to model OCL-constraint evolution	12
C3	A way to obtain an evolution specification	9,10
C4	A way to describe when a model is valid for a given DSL	1,3,4,5, 7,8
C5	A way to describe model co-evolution	13,14
C6	A way to determine if a given co-evolution specification is total	6
C7	A way to derive a co-evolution specification from an evolution specification	10,11, 13

a new version (3). Consequently, models (4) conforming (5) to that DSL (1) should co-evolve (6) into models (7) conforming (8) to the new version of the DSL (3). Since manual co-evolution of models is too costly, we aim to (partially) automate the co-evolution by providing a specification (13) of how models should co-evolve (6) in a dedicated formalism (14). To ease the creation of a co-evolution specification (13), we wish to use a DSL evolution specification (10) to derive (11) a partial co-evolution specification (13).

We stress that one can neither expect the automation nor the co-evolution specification to be *complete*. Theoretical limitations of the approach are discussed in Section 5.

To support the architecture in Figure 4, one has to implement components listed in Table 1. First of all, a formalism is required to model DSL evolution. This formalism can be further decomposed into two components: **C1**, a way to model meta-model evolution (12), and **C2**, a way to model OCL-constraint evolution (12). Furthermore, once the formalism has been chosen a separate component **C3** should focus on obtaining an evolution specification (9,10), *e.g.,* by inspecting previous changes [29].

In order to co-evolve models in response to DSL evolution, one needs **C4**, a mechanism deciding whether model (4,7) is well-formed (5,8) for a given DSL (1,3), as well as **C5** a way to describe model co-evolution (13,14).

Once **C4** and **C5** are available, one needs to determine whether all models would be correctly co-evolved by the co-evolution specification, with respect to the evolution of DSLs. As most DSLs only formally define syntax, and not semantics, this is not a question that can be answered. Hence, we merely require presence of a component, **C6**, capable of determining whether every (syntactically) valid artifact for DSL version 1 can be co-evolved with respect to a co-evolution specification to a (syntactically) valid artifact for DSL version 2. Finally, co-evolution specification depends on the evolution specification, *i.e.,* we need the component, **C7**, presenting a way to do we derive (11) a co-evolution specification (13) from an evolution specification (10).

4. EXISTING WORK

In the literature, several approaches exist that implement (part) of the architecture presented in Figure 4. As mentioned above, although approaches exist for co-evolving transformations [21, 29, 20] and editors [22], we focus on DSL/model co-evolution, as models constitute the vast majority of artifacts in the ASML ecosystems. In the remainder of this section, we discuss previous approaches to DSL/model co-evolution and map them onto our architecture. In this way we assess the state-of-the-art and identify directions for further research. Our discussion of the previous work is indebted to earlier surveys [30, 13].

The approaches we survey (primarily) target EMF-based DSLs. Additional ways to construct DSLs exist [31], with the corresponding ways of dealing with co-evolution.

Specifying Evolution: (10,12).

To allow for the specification of meta-model evolution (10), a formalism is required to capture the evolutionary steps of a meta-model from its original to its evolved version. To this extent we have computed a complete library of atomic evolutionary steps based on the meta-meta-model [32]. Using this library, every sequence of evolutionary steps can be described allowing the implementation of **C1**.

Alternatively, a generic model-to-model transformation language such as QVT [33, 34], or a meta-model independent difference DSL such as EMFCompare [35] can be used.

However, to the best of our knowledge, for OCL (**C2**) a dedicated formalism is still needed.

Obtaining an Evolution Specification: (9,10).

There are several ways of obtaining an evolution specification, which can be divided into two categories: automatic evolution specification approximation and manual evolution specification specification.

Automatic evolution specification approximation is also known as *differencing*. Such approaches such as EMFMigrate [29] and EMFCompare [35] compare the original and evolved meta-model to extract a specification of the difference. A known shortcoming of these approaches is the inability to choose between several evolution specifications that can lead from the original meta-model to the evolved one. For instance, when renaming a class, an identical result may be achieved by deleting the old class and creating a new class. Rose *et al.* [30] argue that no differencing approach can always choose the correct evolution specification.

Manual Evolution Specification Specification can be achieved by means of a predefined collection of operators, or by record-

ing changes. Operator-based methods [14] specify evolution by means of operators, that each encode a (frequently-occurring) pattern of co-evolution. The applicability of this approach relies heavily on the available operators. The state-of-the-art operator-based tool is Edapt[2] [36], which we have evaluated [37], and improved [38]. Change recording approaches record the actions performed on the meta-model by the user in order to obtain an evolution specification. This mitigates the problem presented by the differencing approach, but only is the user works in the correct way. That is, if a user deletes and adds a class, did they indeed intend a delete and add, or did they really intend a rename?

Summarizing, various approaches can be found in the literature to implement **C3**.

DSL/Model Conformance: (5,8).

When a DSL evolves, we can wonder which models are valid before the evolution, and which models are valid after the evolution.

Schoenboeck *et al.* have formalized conformance into a number of OCL constraints [39]. However, this work does not cover all constraints used by modern meta-modeling frameworks such as EMF [4]. González *et al.* have created a transformation from EMF (including OCL constraints) to a constraint solving language CSP [40]. This CSP specification then precisely describes all valid model instances. Lastly, Anastasakis *et al.* have created a similar transformation that formalizes a UML diagram [41] into Alloy [42].

Summarizing, the existing formalisms allow one to describe the set of all valid models for a given meta-model, implementing component **C4**.

Specify Co-Evolution: (13,14).

For the specification of co-evolution, several mature languages/tools exist, the most generic being QVT [33, 34]. Specifically for the co-evolution of models, a tool called Flock [43] exists. We thus consider **C5** to be adequately addressed by these existing tools.

Valid Co-Evolution Specification: (4,6,7,13).

Intuitively, a co-evolution specification takes a model in the original language as input and yields a semantically equivalent model in the evolved language as output. However, DSLs defined by a meta-model and OCL constraints merely define syntax of the models. Hence, addressing validity either calls for analysis of semantics specified elsewhere, or for redefinition of validity in terms of syntax.

Unfortunately semantics are often not formally specified, but embedded in code generators and interpreters. Hence, we relax the notion of validity and require the co-evolution specification to be total, *i.e.,* every valid model for the original DSL is mapped to a valid model in the evolved DSL.

Summarizing, we observe that to the best of our knowledge, no implementation for **C6** is available.

Deriving a Co-Evolution Specification: (10,11,13).

Many of the existing solutions for DSL/model co-evolution, derive a co-evolution specification from the corresponding evolution specification [36, 29].

Additionally, Kappel *et al.* [16] have defined a method called "Model Transformation By-Example". The method starts with the user implementing a co-evolution for a couple of models. From these sample co-evolution specifications, they derive a generic co-evolution specification that should be applicable to all models, dropping the need for an evolution specification all together.

However, regardless of the technique chosen, derived co-evolution specification should be total. Otherwise, a co-evolution specification derived might turn out to be useless. Since no implementation of **C6** is available to the best of our knowledge, we consider implementation of component **C7** to be an open question. Additionally, with respect to this derivation, there are other limitations to consider, which we will discuss in Section 5.

[2]Previously known as COPE

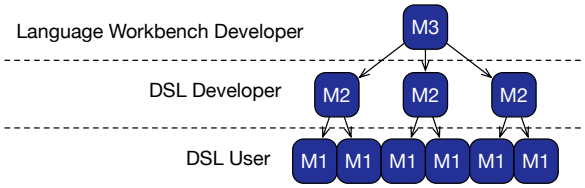

Figure 5: The three levels of meta-modelling, and their available information.

Conclusion.

Having summarized the state-of-the-art for DSL/model co-evolution, we conclude that the following questions should be answered, before the DSL/model co-evolution architecture can be completely implemented:

1. **C2**: How do we specify OCL evolution?
2. **C6**: Is a given co-evolution specification total?
3. **C7**: How do we derive a total co-evolution specification from an evolution specification?

5. THEORETICAL LIMITATIONS TO AUTOMATION

In the previous sections, we have defined a number of components that have to be implemented in order to automate the co-evolution of models with respect to DSL evolution.

Herrmannsdörfer *et al.* have already argued that, in general, no co-evolution specification can fully automate model co-evolution [44]. In the same work, the authors present a solution based on user interaction. However, there are additional limitations not related to user interaction, but related to the information available.

Deriving co-evolution specifications from evolution specifications is hindered by presence of OCL constraints in DSLs. The OCL constraints are not known by the developer of (co-)evolution tooling, as they reside at the level of actual meta-models. We structure the information available to developer of co-evolution tooling by explain the three different levels of information (illustrated in Figure 5).

At **level 1** reside the *Language Workbench Developers*. They have knowledge of a specific meta-meta-model (*e.g.,* `Ecore` [45]), but have no knowledge of specific meta-models (*e.g.,* as the workbench they develop may be used outside their own company). The challenge they face is that any tools, or techniques developed must be generic and applicable to any meta-model conforming to the meta-meta-model.

This means that a researcher wanting to create a complete and reusable piece of evolution tooling must account for every possible meta-model, and every combination of possible OCL constraints on that meta-model. With respect to co-evolutions for these evolutions, every valid instance of that DSL (*i.e.,* any possible meta-model with any valid combination of OCL constraints) should be accounted for. We deem that creating reusable pieces of tooling at this level is, therefore, unfeasible. What remains is to assist the developers at the next level in creating good co-evolution specifications.

At **level 2** reside the *DSL Developers*. DSL developers have knowledge of their own specific meta-model including the OCL constraints present. Additionally the DSL developer has access to the evolution specification of that meta-model and its OCL constraints. However, the DSL developer may still have no knowledge of which instances (models) of their DSL actually exist (*e.g.,* because the models are made at external companies), and must thus (in their work) account for all possible models. However, tooling exists to give a formal definition of what a valid model is (*e.g.,* using EMFtoCSP [40]). In this sense, we could help the DSL developers gain insight into the models that could exist

At the lowest level, **level 3**, reside the DSL users. These users actually create models. At this level there is knowledge of all levels (models, meta-models and meta-meta-models) and the evolution of meta-models. However, creating tooling here is not feasible, as it would have to be re-created for every individual version of every individual DSL.

As, at **level 1** there is not enough information to create static reusable pieces of (co-evolution) knowledge. One can never give a fixed mapping from an evolution specification to a co-evolution specification that works for every meta-model, because for every such mapping there is a possible OCL constraint that can contradict the mapping.

A solution would be to create a function that, given a DSL (meta-model + OCL) and an evolution specification, yields a co-evolution specification. Such a function would have to account for every possible meta-model, and every possible combination of OCL constraints on that meta-model. At present, we do not see how to approach this problem.

We thus believe that creation of the evolution-to-co-evolution mapping should be carried out at **level 2** (DSL developer), as the DSL developers do have knowledge of which OCL constraints should be accounted for. The next obvious goal should thus be to support the DSL developer in creating a valid mapping (*e.g.,* by creating counter examples of models that are not validly co-evolved for a given mapping).

6. CONCLUSION AND FUTURE WORK

In this paper, we have presented a high-level architecture for implementing a tool that automates model co-evolution with respect to DSL evolution to the highest degree possible. Based on this architecture, we have presented a number of questions that have to be answered before the architecture can be fully implemented.

Furthermore, we have looked at the state of the art in DSL/model co-evolution to assess to what extent the posed questions have already been answered. We concluded that specification of both meta-model evolution, and model co-evolution are supported by existing formalisms, but the specification of OCL evolution is not. Furthermore, we observe that there is no formal check whether a given co-evolution specification is valid, and that this question has to be answered before co-evolution specifications can be derived from evolution specifications.

As future work, we consider formal modeling of co-evolution specifications and checking whether their validity. Additionally, we are considering specification of OCL evolution as future work.

7. REFERENCES

[1] 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.

[2] 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.

[3] M. Herrmannsdörfer, D. Ratiu, and G. Wachsmuth, "Language evolution in practice: The history of GMF," ser. LNCS, vol. 5969. Springer, 2009, pp. 3–22.

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

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

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

[7] "OMG SysML," http://www.omgsysml.org/, accessed: 2016-07-05.

[8] M. Lungu, "Towards reverse engineering software ecosystems," in *ICSM*, 2008, pp. 428–431.

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

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

[11] J. Businge, A. Serebrenik, and M. G. J. van den Brand, "An empirical study of the evolution of eclipse third-party plug-ins," in *IWPSE-EVOL*, A. Capiluppi, A. Cleve, and N. Moha, Eds. ACM, 2010, pp. 63–72.

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

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

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

[15] 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.

[16] 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.

[17] L. M. Rose, A. Etien, D. Mendez, D. S. Kolovos, F. A. C. Polack, and R. F. Paige, "Comparing Model-Metamodel and Transformation-Metamodel Co-evolution," in *Model and Evolution Workshop*, 2010.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] D. Di Ruscio, L. Iovino, and A. Pierantonio, "Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems," in *Graph Transformations*, ser. LNCS. Springer, 2012, vol. 7562, pp. 20–37.

[24] 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

[25] F. M. Haney, "Module connection analysis: A tool for scheduling software debugging activities," in *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*, ser. AFIPS '72 (Fall, part I). ACM, 1972, pp. 173–179.

[26] S. S. Yau, J. S. Collofello, and T. MacGregor, "Ripple effect analysis of software maintenance," in *Computer Software and Applications Conference, 1978. COMPSAC '78. The IEEE Computer Society's Second International*, 1978, pp. 60–65.

[27] T. Zimmermann, V. Dallmeier, K. Halachev, and A. Zeller, "erose: guiding programmers in eclipse," in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2005, pp. 186–187.

[28] R. Jongeling, "Change Impact Analysis in Model Driven Software Engineering Ecosystems," Master's thesis, Eindhoven University of Technology, the Netherlands, 2016. [Online]. Available: http://alexandria.tue.nl/extra1/afstversl/wsk-i/Jongeling_2016.pdf

[29] J. Di Rocco, L. Iovino, and A. Pierantonio, "Bridging state-based differencing and co-evolution," in *Workshop on Models and Evolution*. ACM, 2012, pp. 15–20.

[30] 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.

[31] S. Erdweg, T. van der Storm, M. Völter, M. Boersma, R. Bosman, W. R. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. D. P. Konat, P. J. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. A. Vergu, E. Visser, K. van der Vlist, G. H. Wachsmuth, and J. van der Woning, *The State of the Art in Language Workbenches*. Springer, 2013, pp. 197–217.

[32] J. G. M. Mengerink, A. Serebrenik, R. R. H. Schiffelers, and M. G. J. van den Brand, "A complete operator library for DSL evolution specification," in *ICSME*, 2016.

[33] "QVT," http://www.omg.org/spec/QVT/, accessed: 2015-04-07.

[34] "QVTo," http://www.eclipse.org/mmt/?project=qvto, accessed: 2015-04-07.

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

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

[37] 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.

[38] J. G. M. Mengerink, A. Serebrenik, R. R. H. Schiffelers, and M. G. J. van den Brand, "Udapt: Edapt extensions for industrial application," in *IT.SLE*, ser. CEUR-WS, 2016.

[39] E. Burger and A. Toshovski, "Difference-based Conformance Checking for Ecore Metamodels," in *Proceedings of Modellierung 2014*, ser. GI-LNI, vol. 225, 2014.

[40] C. A. González, F. Büttner, R. Clarisó, and J. Cabot, "Emftocsp: A tool for the lightweight verification of emf models," in *Formal Methods in Software Engineering: Rigorous and Agile Approaches*, ser. FormSERA '12. IEEE, 2012, pp. 44–50.

[41] "UML," http://www.uml.org/, accessed: 2016-06-28.

[42] D. Jackson, *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[43] 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.

[44] 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.

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