# Model-Driven Co-Evolution for Agile Development

J. Schönböck
Upper Austrian University of Applied Sciences, Hagenberg, Austria
Email: johannes.schoenboeck@fh-hagenberg.at

J. Etzlstorfer, E. Kapsammer, A. Kusel, W. Retschitzegger, and W. Schwinger
Johannes Kepler University Linz, Austria
Email: {firstname.lastname}@cis.jku.at

***Abstract*—In agile Model-Driven Engineering, the evolution of diverse software artifacts is omnipresent. Especially the evolution of metamodels, defining the grammar of, e.g., Domain Specific Languages, is quite challenging, since many dependent artifacts, including models and transformations, have to be co-evolved to re-establish consistency. Although much research effort has been spent to automate co-evolution, crucial issues remain open and a systematic survey of the state of research across different domains is still missing. This paper provides an extensive survey evaluating various co-evolution approaches also from areas in software engineering like data, ontology, and grammar engineering on basis of a detailed set of criteria serving as a research roadmap for further developments in the area of co-evolution for agile MDE. Based on these results, a conceptual co-evolution framework is presented and illustrated by a running example especially targeting a decrease in co-evolution effort, an increase in co-evolution consistency, and an advance in extensibility.**

## I. Introduction

To cope with the ever increasing complexity of software systems, Model-Driven Engineering (MDE) proposes an extensive use of models to conduct software development on a higher level of abstraction [5], followed by a systematic *transformation* of these models into code, in order to improve the efficiency and effectiveness of the overall system development [1]. Consequently, MDE considers models no longer as mere blueprints as often done in traditional software engineering, but employs models as a more abstract view on the source code. Analogous to the trend of agile software development, agile MDE[1] has emerged recently. Thereby, the main idea is to model not the whole software system at once but to rather continuously evolve the models depending on the users' needs, making agile MDE a technique that perfectly fits into agile process frameworks as e.g., Scrum. Thus models are used from the very beginning of development, e.g., evolutionary prototyping, and continued to be used throughout the whole software development process. Consequently, a proper support for evolution is indispensable in agile MDE.

Whereas changes in the models may be handled, e.g., by re-running a code generator, changes in the underlying *metamodels*, which represent the *grammar* of a modeling language [48], exhibit special challenges due to its crucial role in MDE. However, metamodels are subject to constant *evolution* [20], [24], [39], due to needs for (i) *adaptation* caused by changing software environments, (ii) *perfection* induced by user requirements, or (iii) *correction* because of errors [31], especially if used for defining Domain Specific Languages (DSLs) as, e.g., AUTOSAR[2], which is employed in the automotive industry. This is especially the case if the DSL is still under development (using agile software process models). The crucial issue arising from metamodel evolution is that a tremendous number of dependent artifacts, e.g., models or transformations, may become *inconsistent*. The challenge is to keep the metamodels and *all* dependent artifacts consistent, i.e., it has to be ensured, that despite evolution of the metamodel, different kinds of relationships remain valid. Thus, an evolution of the metamodel inevitably requires automatic *co-evolution* of dependent artifacts to successfully employ agile MDE (cf. Fig. 1).

Although much research effort has been spent to automate co-evolution, crucial issues remain open and a systematic survey of the state of research is still missing. Consequently, this paper provides first, an extensive survey evaluating a variety of co-evolution approaches also from areas in traditional software engineering like data, ontology, and grammar engineering on basis of a detailed set of criteria. This survey is intended to serve also as a research roadmap for further developments in the area of co-evolution for agile MDE by identifying strengths and shortcomings of existing approaches. Second, based on these results, a conceptual co-evolution framework is presented and illustrated by means of a running example especially targeting a decrease in co-evolution effort, an increase in co-evolution consistency, and an advance in extensibility.

The structure of this paper is as follows. Section II presents challenges in metamodel co-evolution building the basis for a detailed evaluation of the current state of research in Section III. Based on the results, Section IV proposes our vision of a co-evolution framework. Finally, Section V concludes the paper by reporting on the expected contributions.

## II. Challenges in Metamodel Co-Evolution

For illustrating evolution from a $\texttt{Metamodel}_{V0}$ to a $\texttt{Metamodel}_{V1}$ and co-evolution of dependent artifacts, Fig. 1 shows an imaginary, but still realistic DSL, used throughout the paper, since despite its simplicity, a number of challenges must be overcome in co-evolving dependent artifacts. During evolution, class `Type` has been extracted from attribute `Car.type` resulting in the classes `Car` and `Type` with the new reference `Car.type` in between. Thus, dependent

This work has been funded by BMVIT under grants FFG BRIDGE 832160 and FFG FIT-IT 825070 and 829598, FFG Basisprogramm 838181, and by ÖAD under grant AR18/2013 and UA07/2013.

[1]http://www.agilemodeling.com

[2]http://www.autosar.org/

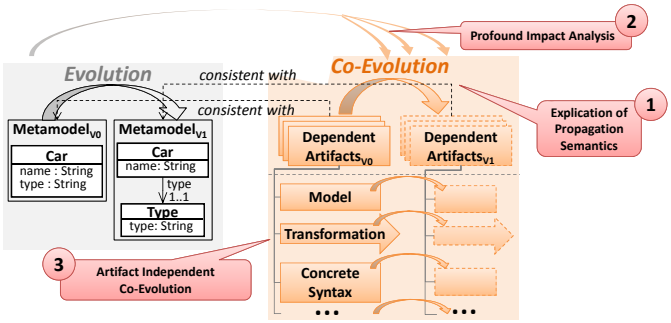


Fig. 1. Challenges in Co-Evolution

artifacts referring to attribute `Car.type` are inconsistent with $\texttt{Metamodel}_{V1}$. The challenges to re-establish consistency are numerous and have been derived on the one hand from literature (cf. Section III) and on the other hand from shortcomings identified by testing dedicated co-evolution approaches.

**Challenge 1: Explication of Propagation Semantics.** A first challenge is to explicitly represent the co-evolution semantics on a high level of abstraction, i.e., model-based (cf. ① in Fig. 1). This is essential since changes on the metamodel do not necessarily determine a specific propagation semantics. For example, for the extraction of class `Type` from attribute `Car.type`, dependent models may either be co-evolved by creating a `Type` instance for each `Car` instance or by creating a `Type` instance for distinct values of the attribute `Car.type`, only. The envisioned explication should be *modular*, to foster reusability and extensibility and should provide means to *adapt* the propagation semantics by the user, independent of a concrete kind of artifact, as also confirmed by an empirical study on the histories of two industrial metamodels in [20]. The explication of the propagation semantics is also key to allow for the co-evolution of diverse dependent artifacts (cf. Challenge 3) and has also been identified in [46].

**Challenge 2: Profound Impact Analysis.** A second challenge is the prediction of unintended effects, e.g., information loss, and propagation costs like manual effort *before* the propagation has taken place, i.e., *impact analysis*, which has been identified as a key feature in co-evolution already in, e.g., [4], [37] (cf. ② in Fig. 1). This is essential in agile processes to check, if the change is worth the expense. An interesting question in the example might be, if the extraction of the class `Type` may be automatically propagated to dependent artifacts or if user-intervention is required. To allow for impact analysis *traceability* between concepts of the metamodel and concepts of the dependent artifacts is required. This traceability, together with the explication of the propagation semantics, should pave the way to *estimate potentially occurring problems*, e.g., changes that cannot be resolved automatically or unintended effects such as information loss, and to estimate the *cost for propagation* in terms of run-time and manual effort.

**Challenge 3: Artifact-Independent Co-Evolution.** Finally, the third crucial challenge is to provide an *artifact-independent* representation of the propagation semantics to ensure consistency between artifacts, e.g., when extracting the class `Type` in the example, models, transformations, and concrete syntax have to be co-evolved following the same semantics (cf. ③ in Fig. 1), as also stated in [39]. Furthermore, the representation of the propagation semantics should be open for different evolution tools (that deliver the actual changes that need to be co-evolved) and also propagation languages (which are capable to actually co-evolve a certain artifact).

## III. State of Research

Evolution in software development has been studied since decades, ranging from Lehman's laws for software evolution [32], to refactoring operations [38], program comprehension [50], or change impact analysis [35] to mention just a few. However, the focussed topic of evolving metamodels and co-evolving dependent artifacts, falling into the category of *coupled software transformations* [28], still raises challenges, that have not been adequately tackled up to now, hindering the successful application of MDE in agile development processes. In the following an extensive survey on the state of research evaluating a variety of co-evolution approaches is presented.

### A. Rationale behind Selection of Approaches

Subsequently, not only related work from the most closely related area of *MDE* is discussed, but also complementary approaches from *data*, *ontology*, and *grammar engineering* are considered to obtain a comprehensive picture of the state of research and to open up the opportunity to potentially adapt promising related approaches for co-evolution in MDE. To achieve comparability, the role of database schemas, XML schemas, ontologies, and grammars is seen analogous to that of metamodels and the role of relational tuples, XML documents, ontology instances, and program code is seen analogous to that of models. Similar analogies may be drawn for other dependent artifacts such as transformations. Relevant approaches have been chosen, in a first step by searching digital libraries for dedicated surveys using the keywords evolution and co-evolution. Additionally, the proceedings of closely related conferences such as MoDELs[3], or ICMT[4] as well as journals such as SoSym[5] of the last 10 years have been searched. Finally, relevant references stated in the found papers have been followed in order to complete the overview on the co-evolution research. Besides numerous approaches, two dedicated surveys ( [19], [44]) have been included in the comparison. However, none of those investigated approaches across different areas. Subsequently, approaches of the diverse engineering domains are investigated in detail with respect to the challenges described above (cf. Table I).

### B. State-of-the-Art in Explication of Propagation Semantics

In a first step, related co-evolution approaches are examined concerning which *mechanisms* are provided to explicate the propagation semantics and to which respect *modularity* and *adaptability* of propagation semantics is considered.

[3]ACM/IEEE International Conference on MDE Languages and Systems
[4]International Conference on Model Transforamtions
[5]Journal of Software and Systems Modeling

TABLE I
OVERVIEW ON EXISTING CO-EVOLUTION APPROACHES

| | | Framework Name or Reference | Explication of Propagation Semantics | | | Impact Analysis | | Supported Artefacts | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Mechanism | Modularity | Adaptablity of Propagation Semantics | Traceability | Problem & Cost Estimation | Models | Transformations | Other Artifacts (Concrete Syntax, Docu, ...) |
| Model Engineering | | [47] | DSL | ✓ | N.A. (User-defined) | × | × | ✓ | × | × |
| | | MCL [40] | DSL | ✓ | N.A. (User-defined) | × | × | ✓ | × | × |
| | | MCL [34] | DSL | ✓ | N.A. (User-defined) | × | × | × | ✓ | × |
| | | Epsilon Flock [45] | DSL | ✓ | N.A. (User-defined) | × | × | ✓ | × | × |
| | | [56] | ATL Refinement Mode | × | N.A. (User-defined) | × | × | ✓ | × | × |
| | | [25] | Graph Transformations | × | N.A. (User-defined) | × | × | ✓ | × | × |
| | | [36] | Graph Transformations | × | ~ Adaptable by User on Graph Transformation Level | × | × | ✓ | × | × |
| | | [15] | ETL | × | ~ Adaptable by User on Code Level | × | × | ✓ | × | × |
| | | [7] | ATL | × | × | × | × | ✓ | × | × |
| | | AML [11] | ATL | × | ~ Adaptable by User on Code Level | × | × | ✓ | × | × |
| | | [6] | Java | × | × | × | × | ✓ | × | × |
| | | [12] | ATL | × | × | × | × | × | ✓ | × |
| | | EMF Migrate [46] | DSL | ~ Change-dependent Libraries | ~ Adaptable by User on DSL Level | ✓ | ~ Adaptation Cost Functions | ✓ | ✓ | ~ GMF models |
| | | [13] | ATL | × | × | × | × | × | ✓ | × |
| | | [54] | (QVT Relations | × | × | × | × | ✓ | × | × |
| | | COPE [21] | Groovy, Java | × | ~ Adaptable by User on Code Level | × | × | ✓ | × | × |
| | | [39] | Graph Transformations | × | ~ Adaptable by User on Code Level | × | × | ✓ | ✓ | × |
| | | [26] | Java | × | × | × | × | × | ✓ | × |
| Data Engineering | DB | PRISM [8] | SQL | × | × | × | × | ✓ | ✓ | × |
| | | HECATAEUS [42] | N.A. (Propagation Not Supported) | × | × | ✓ | ~ Visual Impact Graph | × | ✓ | × |
| | | MeDEA [10] | SQL | × | × | × | × | ✓ | × | × |
| | XML | X-Evolution [17] | DSL | × | ~ Adaptable by User on DSL Level | ~ | ~ Invalid Documents are Highlighted | ✓ | × | × |
| | | XML Evolution Manager (XEM) [51] | unknown | × | × | × | × | ✓ | × | × |
| | | [29] | XSLT | × | × | × | × | ✓ | × | × |
| Ontology Engineering | | KAON [49] | unknown | × | × | × | × | ✓ | × | ✓ |
| | | Protégé [41] | N.A. (Propagation Not Supported) | × | × | × | × | × | × | × |
| Grammar Engineering | | TransformGen [14] | DSL | ✓ | ~ Adaptable by User on Code Level | × | × | ✓ | × | × |
| | | Lever [43] | DSL | ✓ | ~ Adaptable by User on Code Level | × | × | ✓ | × | × |

**Mechanism.** In *MDE*, the vast majority of approaches supports the actual propagation of changes [6], [7], [11]–[13], [15], [21], [25], [26], [34], [36], [39], [40], [45], [47], [54], [56], while a single approach [13] aims at helping the evolution designer in the co-evolution process by annotating the possibly affected elements that need to be co-evolved. However, only five approaches [34], [40], [45]–[47] provide *mechanisms* to explicate the propagation semantics by allowing the evolution designer to access the propagation semantics on basis of a DSL. However, although these explicate the propagation semantics, the underlying semantics is typically hidden in a low-level co-evolution script and can not be adapted by the user, i.e., the DSL provides no means for extensibility and adaptability (cf. below). The remaining approaches base on the use of ordinary transformation languages.

Regarding *relational databases*, support for the *explication of the propagation semantics* is not provided. Instead, the propagation semantics are tightly coupled to the evolution steps, which are typically described in SQL. In HECATAEUS co-evolution is merely hypothetical, i.e., support for the actual propagation is not provided. In the area of *XML schema evolution*, no comprehensive co-evolution approach has been developed so far. Thus, current approaches concentrate on the provision of change operators for the evolution of DTDs (cf., e.g., XEM [29], [51]) or XML schemas (cf., e.g., X-Evolution [17]) and the co-evolution of XML documents as dependent artifacts. Thereby only X-Evolution provides means to explicate the propagation semantics in terms of a DSL.

In the field of *ontology evolution*, the KAON framework [49] proposes an approach consisting of four phases, being (i) change representation, (ii) determination of the semantics of a change, (iii) implementation, and (iv) propagation. The first two phases focus on the representation of the actual evolution of the ontology, only. However, only the changes but not the actual co-evolution semantics is made explicit to the user. In contrast to KAON, other approaches such as Protégé [41] typically concentrate on the evolution of the ontology, only but do not provide means for co-evolution of dependent artifacts.

Concerning *grammar engineering*, the Lever approach [43] and the TransformGen approach [14] support the *explication of the propagation semantics* on basis of dedicated DSLs.

**Modularity of Propagation Semantics.** To allow for reusability and composition of propagation semantics, which is crucial when dealing with various artifacts, modularity is of great interest. Although many of the approaches in the area of *MDE* distinguish between atomic and composite changes on the metamodel and thus, allow for modularity on the level of changes, the propagation semantics is typically specified in a monolithic manner. Only some approaches [34], [40], [45],

[47] provide basic modularity mechanisms such as functions or facilities to create libraries of adaptation rules for each kind of dependent artifact, such as EMFMigrate [46]. These adaptation rules may build on each other by two dedicated keywords, being `refine`, which allows to describe the sequential execution of rules, and `replace`, which allows to overwrite the propagation semantics of an existing adaptation rule. Each adaptation rule, however, is bound to a specific change. Consequently, reuse of propagation semantics across the boundaries of changes is not supported.

Regarding *relational databases*, none of the investigated approaches support modularity of propagation semantics. In the area of *XML schema evolution*, in [18] a set of atomic evolution operations for XML schemas is presented, which allows for composition as employed in X-Evolution [17]. For the XEM system [29], [51], a set of DTD change primitives is proposed. In either case, modularity of propagation semantics is not an issue.

In *ontology engineering*, the KAON framework [49] provides modularity by allowing to combine simple evolution strategies to so-called advanced evolution strategies, enabling to control the evolution by specifying general goals such as minimizing the number of ontology changes. Due to missing support of the change propagation phase, Protégé [41] does not support modularity.

Regarding *grammar engineering*, the TransformGen approach [14] follows the same principles as operation-based approaches in MDE, i.e., approaches, which tightly couple each evolution step on the metamodel to corresponding operations on the dependent artifact. The generated transformation builds on modular and composable parts. In Lever [43], the evolution language builds on modular parts and is extensible, allowing to add own semantics.

**Adaptation of Propagation Semantics.** A major issue in agile *MDE* is the adaptation of the propagation semantics for dependent artifacts, since changes on the metamodel do not necessarily determine a single specific propagation semantics for dependent artifacts. Consequently, approaches to metamodel evolution should at least allow to overrule a given default semantics by some user-specified propagation semantics or even better provide a choice of predefined propagation semantics to the evolution designer. In this respect, approaches which employ a manual specification of the propagation semantics [25], [34], [40], [45], [47], [56], may not be classified according to this criterion, since the propagation semantics is user-defined anyway. Concerning the remaining approaches, only six of them [11], [15], [21], [36], [39], [46] allow to overrule a given default propagation semantics. However, adaptation of the propagation semantics can be done at the code level, only, i.e., dedicated adaptation mechanisms on a higher level of abstraction are missing.

Regarding *relational database schemas*, PRISM [8] and MeDEA [10] do not allow for adaptation. In HECATAEUS [42], the only adaptation possibility refers to the decision whether or not to propagate a certain change to a particular artifact. Concerning *XML schema evolution*, only X-Evolution [17] allows to adapt the propagation semantics on the code level, whereas the remaining approaches do not allow for such facilities.

In *ontology engineering*, by the provision of resolution points and resolution strategies, the evolution designer is able to adapt the effects of changes within the ontology in the KAON framework [49], but not the propagation semantics. This means that the user is allowed to choose different co-evolution strategies, so-called resolution points, for which different default evolution strategies are provided. The list of the resolution points is then presented to the evolution designer allowing to approve the changes before the actual propagation. Due to missing support of change propagation, Protégé [41] does not allow for adaptability.

In *grammar engineering*, TransformGen [14] allows to customize the co-evolution on basis of user-written functions, i.e., on the code level. In Lever [43], the DSL may not be adapted, but only the generated code.

### C. State-of-the-Art in Impact Analysis

Second, related work with respect to impact analysis is discussed, focussing on the establishment of *traceability* and the estimation of arising *problems and costs* in co-evolution.

**Traceability.** When surveying existing approaches in the field of *MDE* concerning traceability between the metamodel and dependent artifacts, solely a single approach provides support [46]. In particular, they propose a dedicated mechanism that allows to establish traceability links between the metamodel and different kinds of artifacts [22]. In [9], this work is extended by allowing to visualize the traceability between the metamodel and the dependent artifacts on basis of TraceVis [2], a tool proposed for the visualization of traceability links between a chain of model transformations. Although the establishment of traceability is supported, the exploitation thereof is left open to a large extent, demanding for further investigation. In particular, changes that cannot be propagated automatically should be especially highlighted, since these changes cause high manual effort in an agile development process.

In the area of *relational databases*, only HECATAEUS [42] enables the evolution designer to visually examine the impact of schema evolution on queries and views as dependent artifacts by means of a visual impact graph. MeDEA [10] supports traceability between a conceptual model such as UML, on which changes are defined, and a logical model such as a relational schema to which changes are propagated, but not between the schema and dependent artifacts. PRISM [8] does not support impact analysis at all. Regarding the domain of *XML schema evolution*, X-Evolution [17] does not allow for explicit traceability, but rather for computing potentially occurring invalidities [18]. In contrast, XEM [51] and [29] do not support traceability at all.

In *ontology engineering* KAON [49] and Protégé [41] do not provide impact analysis and thus, also no traceability.

In *grammar engineering*, both approaches do neither support impact analysis nor traceability.

**Problem and Cost Estimation.** As may be concluded from the discussion above, impact analysis for supporting agile *MDE* is still in its infancy. Since being at the very beginning, the estimation of potentially arising problems in dependent artifacts as, e.g., information loss or non-resolvable changes, is not fully targeted by any approach yet. However, this will be the next logical step after having successfully established traceability between the metamodel and dependent artifacts as mentioned in [22]. Similar to the estimation of potentially occurring problems in dependent artifacts, the estimation of the cost for propagating changes to dependent artifacts in terms of runtime and manual effort needed is still poorly understood. First work in this direction has been published recently providing an estimation of costs in the context of propagating changes to dependent model transformations [46]. In this work, the costs for adapting ATL transformations are estimated on basis of dedicated adaptation cost functions. However, providing a single number to accumulate potentially occurring problems and costs is a first step, only.

In the domain of *relational databases*, PRISM [8] does not provide any support for problem and cost estimation. In contrast, HECATAEUS [42] highlights all affected elements of dependent artifacts visually, but is not able to provide a detailed problem and cost report. MeDEA [10] does not support any problem and effort estimation. In the area of *XML schema evolution*, only X-Evolution [17] visually points out all documents, which will not remain valid due to a given change together with all the invalid elements, thereby guiding the evolution designer in the resolution process.

In *ontology engineering*, KAON [49] and Protégé [41] do neither consider the phase of impact analysis which is also true for both approaches considered in *grammar engineering*.

### D. State-of-the-Art in Supported Artifacts

Finally, related work is examined to which extend different kinds of dependent artifacts may be co-evolved.

Regarding the area of *MDE*, the majority of existing co-evolution frameworks focuses on a specific kind of dependent artifact to be co-evolved, being mostly models [6], [7], [11], [15], [21], [25], [36], [40], [45], [47], [54], [56]. First approaches for the co-evolution of transformations are presented in [12], [13], [26], [34]. Currently, solely EMFMigrate [46] and Meyers et. al [39] target the propagation of changes to several kinds of dependent artifacts. However, they still require the specification of an artifact-specific co-evolution semantics.

Regarding *relational databases*, relational tuples are the primary artifact to be co-evolved and supported by PRISM [8] and MeDEA [10]. Additionally, PRISM allows to co-evolve SQL queries. HECATAEUS [42] focuses on queries and views as dependent artifacts, but for impact analysis, only. Again specific solutions are built for specific kinds of dependent artifacts, i.e., no approach may be considered generic. In the field of *XML schemas*, XML documents are considered as the only artifacts to be co-evolved (cf., e.g., XEM [29], [51], and X-Evolution [17]), i.e., artifact-independence is not an issue.

In *ontology engineering*, the KAON framework [49] considers instances as dependent artifacts. Dependent ontologies and applications are mentioned, but not further dealt with. In Protégé [41] propagation to dependent artifacts is not an issue.

Concerning *grammar engineering*, both approaches, TransformGen [14] and Lever [43], provide support for the co-evolution of program code, only.

### E. Reflection of Related Work

In summary, although numerous co-evolution frameworks have been proposed in different domains, a generic framework, i.e., independent of a certain kind of dependent artifact, comprising an explication of the propagation semantics, means for impact analysis and change propagation as urgently needed for agile MDE is still missing. Despite the fact that first approaches offer dedicated DSLs for the explication of the propagation semantics, none of them abstracts from the kinds of dependent artifacts, but typically focus on a specific one, only. Additionally, modularity and adaptable propagation semantics is supported by few approaches, only and most often restricted to the code level. Thus, research efforts to provide modularity and adaptability on a higher level of abstraction are needed for agile MDE. As argued above, impact analysis in MDE and in related engineering domains has not gained much momentum so far. Although some groundwork in the establishment of traceability between a metamodel and dependent artifacts has been performed, the interpretation thereof in terms of estimating potentially arising problems or costs incurring for propagation is still in its infancy, emphasizing the need for further research. This is in contrast to the area of software engineering, where much research on impact analysis of code changes has been investigated as a recent survey reveals [33]. Although not being directly applicable to the domain of agile MDE, many fruitful ideas may be gained from techniques in software engineering. Thus, in the following a framework called CoEvolver will be presented that tackles these limitations. The core idea is to present a DSL that allows to model the propagation semantics of either fine-grained atomic changes (add, remove, delete, move) or more complex composite changes. However, the propagation semantics of these composites changes may be derived by the combination of the propagation semantics of the fine grained changes, overcoming the current shortcomings in the explication of the propagation semantics.

## IV. Conceptual Co-Evolution Framework

After having discussed the state-of-the-art in co-evolution in detail, the following section outlines a conceptual framework called *CoEvolver*, which tackles the afore mentioned limitations and provides the basis for an actual implementation in an agile MDE environment (cf. Fig. 2).

*CoEvolver* supports three core phases in the co-evolution process, being (i) the explication of the propagation semantics on basis of a visual DSL, (ii) the impact analysis, and (iii) the propagation of the changes to dependent artifacts. The support of the metamodel evolution itself is out of scope of

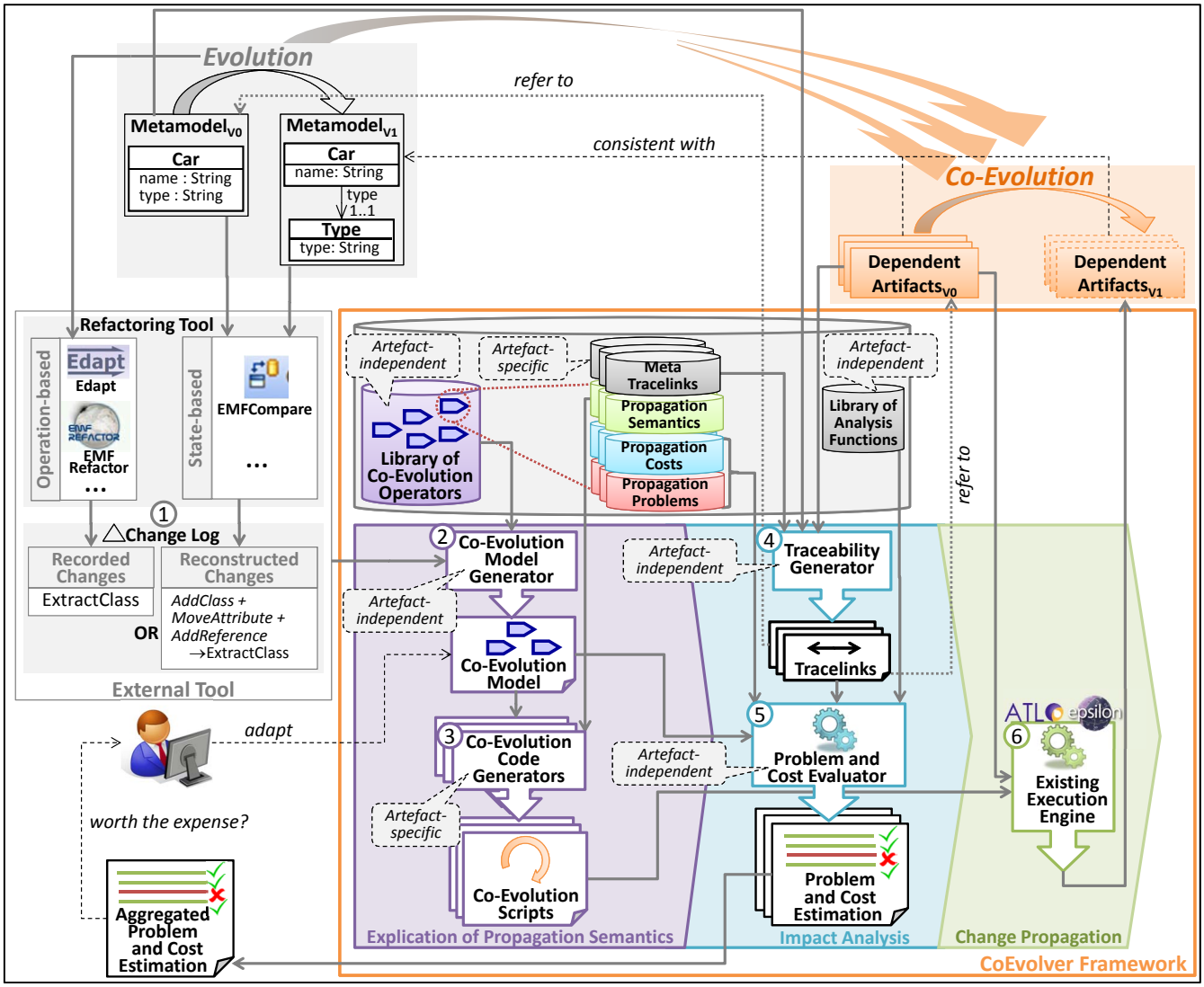


Fig. 2. Overview on the *CoEvolver* Framework by Phases

the *CoEvolver* framework, since this might be conveniently performed by existing metamodel editors or refactoring tools such as EMF Refactor [3] or EMF Compare[6], which are intended to use CoEvolver as a target platform for co-evolution. At design-time, *CoEvolver* is configured with a library of co-evolution operators, meta tracelinks (i.e., tracelinks between the involved metamodels), propagation semantics, costs, problems per dependent artifact, and a library of analysis functions. At run-time, first, the co-evolution model generator (cf. ② in Fig. 2) takes the metamodel changes as input (cf. `ExtractClass` in the running example), which are either recorded, i.e., operation-based, or reconstructed, i.e., state-based, depending on the tool used for refactoring (cf. ① in Fig. 2). As an output, it automatically generates an explicit representation of the propagation semantics on basis of the *co-evolution DSL*, i.e., a co-evolution model (cf. Section IV-A) based on the library of pre-defined co-evolution operators. After having derived the co-evolution model, the evolution designer may adapt the propagation semantics to her needs by dedicated components of the DSL on a model level.

Besides the propagation semantics, each component of the DSL exhibits potentially arising propagation problems and propagation costs, which are the basis for *impact analysis* (cf. Section IV-B). For this, first traceability between the metamodel in its original version and the dependent artifacts must be established by a traceability generator (cf. ④ in Fig. 2) with the help of meta tracelinks. This builds the basis for the problem and cost evaluator (cf. ⑤ in Fig. 2) for deriving a report with the help of a library of analysis functions. If the calculated costs fulfill the evolution designer's expectation, i.e., are worth the expense, the actual changes may be propagated to dependent artifacts in the third phase, being the *change propagation* itself.

To achieve the co-evolution (cf. ⑥ in Fig. 2) according artifact-specific co-evolution code generators (cf. ③ in Fig. 2) are needed, which translate the specified propagation semantics into artifact-specific co-evolution scripts in a

[6]http://www.eclipse.org/emf/compare/

dedicated propagation language such as Epsilon Flock [45] or a general purpose transformation language such as ATL [23]. The actual execution of the change propagation re-establishes consistency between the dependent artifacts and the evolved metamodel, e.g., re-establishment of the "conforms to" relationship between the models and the metamodel. Since *CoEvolver* describes the propagation semantics on basis of a single co-evolution model, a consistent co-evolution across diverse kinds of artifacts may be achieved.

### A. Explication of the Propagation Semantics

One main idea of the *CoEvolver* framework is to provide a DSL for the explication of the propagation semantics to enable a model-driven approach for the co-evolution of dependent artifacts in agile MDE (cf. Fig. 3). Regarding the example in Fig. 3, the refactoring operation `ExtractClass` has been applied. Consequently, both versions of the metamodel manifest the same semantics, while exhibiting the structural difference that attribute `Attribute.type` has been made explicit by means of class `Type` in the evolved version of the metamodel (cf. ① in Fig. 3). Such differences are called structural heterogeneities [30], i.e., the expression of semantically similar concepts by means of different metamodeling concepts. Since structural heterogeneities play a distinctive role in metamodel evolution, a co-evolution DSL must be able to cope with them accordingly. For providing reusable components for the resolution of structural heterogeneities in the area of model transformations we developed a so-called Mapping Operator (MOp) language [27], [57] on basis of a systematic classification of heterogeneities [55], covering the structural differences which potentially occur between Ecore-based metamodels. Consequently, in a first step we envision to extend and adapt the MOps to the domain of metamodel evolution, e.g., dealing with additions and deletions of metamodel elements which are not required in a transformation scenario, resulting in a library of *Co-Evolution Operators* (CoEvOps) (cf. ② in Fig. 3). The rationale behind is to raise the level of abstraction to foster an agile process by following an MDE-based approach to co-evolution, since CoEvOps may be seen as independent wrt. both, dependent artifacts and employed propagation language. This abstract representation may then be automatically translated to different propagation languages by means of higher-order transformations (HOT) [52].

**Modularity.** CoEvOps may be divided into a fixed set of fine-grained *kernel* CoEvOps such as an $A_2C$ (Attribute-to-Class) and an extensible set of more coarse-grained *composite* CoEvOps such as a `CoCreateClass`. Kernel CoEvOps are intended to provide the necessary expressivity on the level of classes, attributes, and references, since they are intended to be systematically derived from potentially occurring heterogeneities between two Ecore-based metamodels, i.e., from the Ecore meta-metamodel[7]. In contrast, composite CoEvOps aggregate a certain set of kernel CoEvOps to more coarse-grained components and thus, operate on a higher level of abstraction, as being closer to the thinking of the co-evolution designer. By this modular composition, essential advantages may be achieved. Above all, it is sufficient to specify the propagation semantics (cf. below) and the impacts (cf. Section IV-B) on the level of kernel CoEvOps, only. Consequently, extensibility on the level of composite CoEvOps is achieved without having to define new propagation semantics or additional problems and cost functions for newly introduced composite CoEvOps. To assemble the presented kernel CoEvOps to composite CoEvOps and to bind them to specific metamodels, every CoEvOps has input ports with required interfaces (left side of the component) as well as output ports with provided interfaces (right side of the component), typed to classes (C), attributes (A), and relationships (R). Since there are dependencies between CoEvOps, e.g., a reference demands a source and a target class, every CoEvOp typed to a class offers a trace port (T) at the bottom of the CoEvOp, providing context information. Since CoEvOps are expressed as components, the user can apply them in a plug & play manner.

*Running Example.* Having a look at the example in Fig. 3, one may see that the intended co-evolution is described with the `CoExtractClass` composite CoEvOp, which composes two other composite CoEvOps, being a `CoCopyClass` and a `CoCreateClass`. These finally build upon dedicated kernel CoEvOps, which determine the propagation semantics on a fine-grained level, e.g., the $A_2C$ kernel CoEvOp states that for each distinct value of attribute `Attribute.type` a corresponding instance of attribute `Type.type` should be created, in case of co-evolving models (cf. ③ in Fig. 3).

**Adaptablity.** The composite CoEvOps are *adaptable*, i.e., they may compose different configurations of CoEvOps, depending on the propagation semantics to be realized. A certain composite CoEvOp contains fixed parts but also optional parts, that allow for user adaptations of the propagation semantics on the model level, instead of just on the code level.

*Running Example.* When considering the example in Fig. 3, it may be seen that the evolution designer may adapt the `CoExtractClass` operator to either apply a *copy* (in terms of a `CoCreateClass` CoEvOp) or a *distinct* semantics (in terms of a `CoCreateDistinctClass` CoEvOp). When considering models as dependent artifacts, this would either result in as many `Type` instances as `Attribute` instances in case of the copy semantics or in as many `Type` instances as distinct values of attribute `Attribute.type` in case of the distinct semantics (cf. ③ in Fig. 3).

**Compilation to Executable Formalism.** To actually execute the co-evolution, artifact-specific co-evolution scripts in a specific propagation language need to be automatically generated with the help of co-evolution code generators on the basis of the kernel CoEvOps (cf. ④ and ⑤ in Fig. 3). As stated in Section III, various propagation languages for the actual co-evolution of dependent artifacts have been proposed, e.g., Epsilon Flock [45]. Thus, we intend to base on existing propagation languages for the compilation of the co-evolution model to a co-evolution script, thereby achieving the advantage of reusing existing execution engines (cf. ⑦ and ⑧ in Fig. 3).

[7] http://www.eclipse.org/modeling/emf

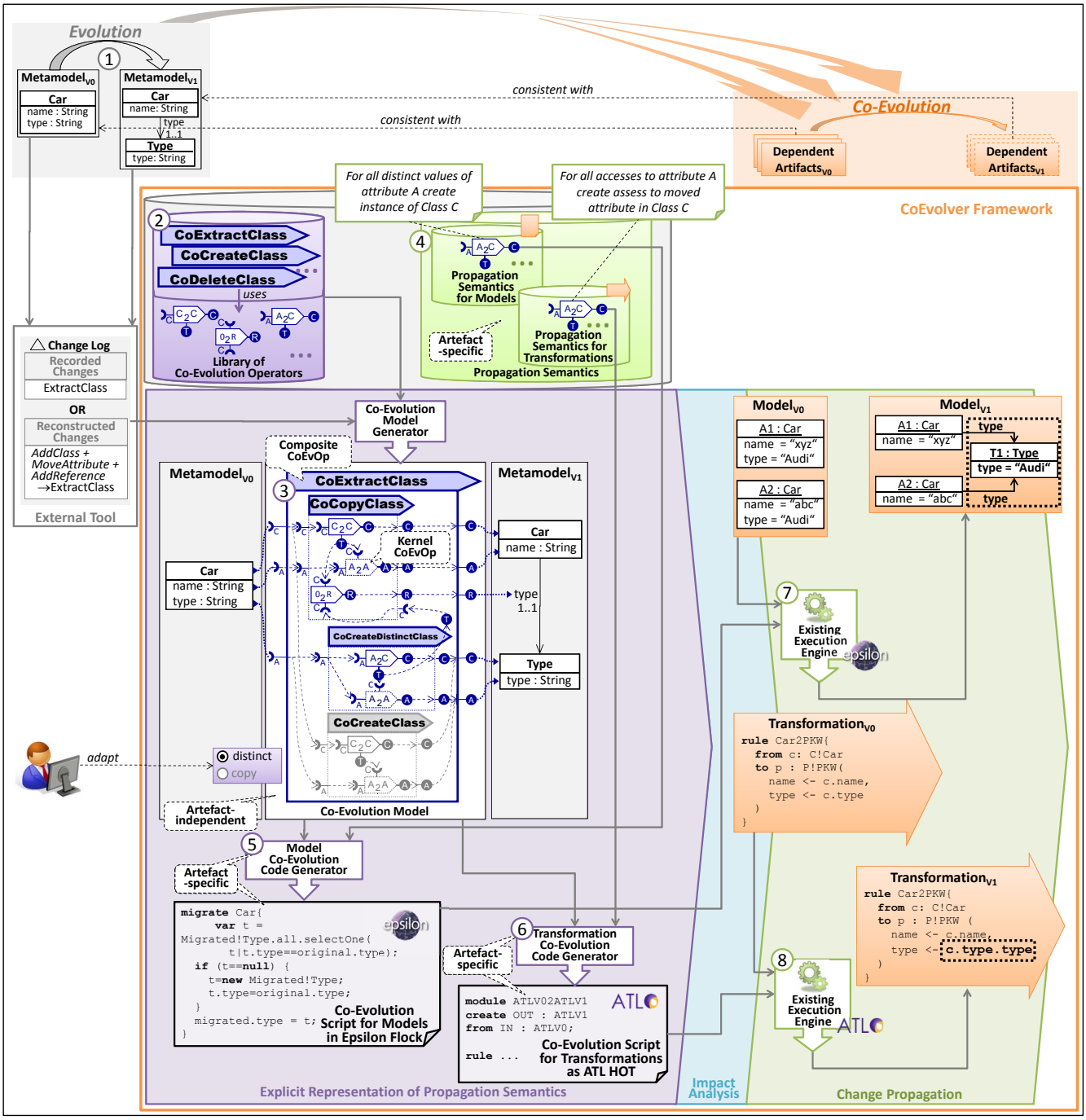


Fig. 3. Explicit Representation of Propagation Semantics and Change Propagation

However, since the focus of current approaches is on models only, we will investigate on strategies for the remaining kinds of dependent artifacts, especially on transformations where we envision to follow a *program transformation* [53] approach, i.e., applying a so-called higher-order transformation (HOT) which rewrites an existing transformation.

*Running Example.* To illustrate a potential outcome of a model co-evolution script, Fig. 3 shows the specified propagation semantics on basis of Epsilon Flock [45]. Furthermore, a HOT is sketched as the potential outcome of a transformation co-evolution script. The goal of this HOT is to rewrite line 5 of the ATL transformation from `type <- a.type` to `type <- a.type.type`, to correctly follow the reference `Attribute.type` and to access the attribute `Type.type`.

### B. Impact Analysis

To determine if the performed changes on the metamodel have a problematic impact on dependent artifacts, an accurate estimation of potentially arising problems and effort needed to propagate changes to dependent artifacts is essential to decide, whether to actually propagate the changes, consider alternative changes or to leave the metamodel as it is. Consequently,

potential problems such as the loss of information that may arise during co-evolution need to be identified and accordingly classified. For this, the kernel CoEvOps need to be classified into categories concerning their effect on dependent artifacts, e.g., *breaking* versus *non-breaking* changes for models as proposed in [54] or according categories for transformations as proposed in [34]. Additionally, adaptable cost functions should be assigned to kernel CoEvOps to be able to calculate the resulting effort in terms of runtime and manual effort. Thereby, costs may vary depending on whether a certain CoEvOp requires the intervention of the evolution designer or leads to a loss of information capacity, e.g., due to deletion. We envision to also provide means for the evolution designer to specify overall goals, e.g., to avoid information loss, leading to automatically adapted co-evolution operators that best meet the specified overall goals, inspired by KAON [49].

**Traceability.** For impact analysis, tracelinks between the metamodel and the dependent artifacts in their original versions need to be established. For being able to automate the establishment of these tracelinks on basis of a dedicated traceability generator (cf. ④ in Fig. 2), meta tracelinks relate elements of the Ecore meta-metamodel to elements of the metamodel of the dependent artifact.

**Problem and Cost Estimation.** To provide means for problem and cost estimation, we envision a library of analysis functions on basis of OCL[8] that allows to conduct queries on the established traceability model. Such OCL queries may allow to ask questions like "How many elements are affected by a certain CoEvOp?" or "Does the CoEvOp cause changes in a dependent artifact that cannot be resolved automatically?". Furthermore, the traceability model may be queried to allow for custom analysis. As a result of the OCL queries, a list of the affected elements per kernel CoEvOp may be achieved, which might get aggregated according to the composition hierarchy. These affected elements act as input to calculate potential problems and costs. As a final result, a profound problem and cost estimation report is provided, allowing to decide whether to propagate the changes or not.

## V. Prototype and Expected Contributions

The omnipresence of evolution in agile MDE and thus, the necessity of proper tool support enabling the co-evolution of arbitrary artifacts at a high level of abstraction was the major motivation behind the conceptualization of *CoEvolver*. We are currently working on the co-evolution DSL on the basis of a graphical Eclipse-based editor as well as the compilation to an executable formalism whereby we focus on the co-evolution of model transformations. Thereby we base on our *Mantra* framework built for applying our MOps as well as for testing and debugging transformations [16]. From these very first results the following contributions may be expected.

**Decreased Effort in Co-Evolution.** First, since the *CoEvolver* framework follows a model-driven approach to the co-evolution of dependent artifacts and since a default co-evolution model is derived automatically from dedicated changes, the effort for co-evolution may be drastically decreased, whereby the MDE based approach allows to adapt the propagation semantics on a model level, relieving the evolution designer from the code level. Concerning scalability, CoEvMops are intended to tackle these problem. Furthermore, we expect rather small evolution steps in agile processes than an extensive redesign of the metamodel, which would trigger complex co-evolution steps.

**Consistent Co-Evolution of Diverse Kinds of Artifacts.** Second, since *CoEvolver* describes the propagation semantics using a single co-evolution model, a consistent co-evolution across diverse kinds of artifacts may be achieved. However, the actual consistency depends on the correct implementation of the co-evolution script code generators. Nevertheless, a single starting point in terms of the co-evolution model offers a solid basis to achieve a globally consistent co-evolution across diverse kinds of artifacts as needed in agile MDE.

**Improved Openness wrt. Artifacts, Refactoring Tools, and Propagation Languages.** Since *CoEvolver* builds on an artifact-independent co-evolution model to describe the propagation semantics, co-evolution of new kinds of artifacts, exhibiting a dedicated metamodel, may be achieved by just implementing dedicated co-evolution script code generators, which realize an interpretation of the propagation semantics for the newly introduced kind of artifact. Furthermore, existing propagation languages may be incorporated by being the target language of a co-evolution script code generator. Finally, CoEvolver is open to existing refactoring tools, since the only interface to them is the change log. Thus, CoEvolver leverages extensibility concerning new kinds of artifacts, as well as reuse of propagation languages and refactoring tools.

[8] http://www.omg.org/spec/OCL

## References


[1] A. Aitken and V. Ilango, "A comparative analysis of traditional software engineering and agile software development," in *In Proc. of46th Hawaii International Conference on System Sciences*, 2013, pp. 4751–4760.

[2] M. Amstel, M. Brand, and A. Serebrenik, "Traceability Visualization in Model Transformations with TraceVis," in *Theory and Practice of Model Transformations*. Springer-Verlag, 2012.

[3] T. Arendt and G. Taentzer, "A tool environment for quality assurance based on the Eclipse Modeling Framework," *Automated Software Engineering*, vol. 20, no. 2, pp. 141–184, 2013.

[4] R. S. Arnold, *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[5] J. Bézivin, "On the Unification Power of Models," *Software & Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[6] M. Brand, Z. Protic, and T. Verhoeff, "A Generic Solution for Syntax-Driven Model Co-evolution," in *Objects, Models, Components, Patterns*. Springer-Verlag, 2011.

[7] A. Cicchetti, D. Ruscio, and A. Pierantonio, "Managing Dependent Changes in Coupled Evolution," in *Theory and Practice of Model Transformations*. Springer-Verlag, 2009.

[8] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful Database Schema Evolution: the PRISM Workbench," *Proc. of the VLDB Endowment*, vol. 1, pp. 761–772, 2008.

[9] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Traceability Visualization in Metamodel Change Impact Detection," in *Proc. of 2nd Workshop on Graphical Modeling Language Development*, 2013.

[10] E. Domínguez, J. Lloret, A. L. Rubio, and M. A. Zapata, "MeDEA: A database evolution architecture with traceability," *Data & Knowledge Engineering*, vol. 65, no. 3, pp. 419–441, 2008.

[11] K. Garcés, F. Jouault, P. Cointe, and J. Bézivin, "Managing Model Adaptation by Precise Detection of Metamodel Changes," in *Model Driven Architecture - Foundations and Applications*. Springer-Verlag, 2009.

[12] K. Garcés, J. M. Vara, F. Jouault, and E. Marcos, "Adapting transformations to metamodel changes via external transformation composition," *Software & Systems Modeling*, vol. 12, no. 1, pp. 1–18, 2013.

[13] J. García, O. Diaz, and M. Azanza, "Model Transformation Co-evolution: A Semi-automatic Approach," in *Software Language Engineering*. Springer-Verlag, 2013.

[14] D. Garlan, C. W. Krueger, and B. S. Lerner, "TransformGen: Automating the Maintenance of Structure-Oriented Environments," *ACM Transactions on Programming Languages and Systems.*, vol. 16, no. 3, pp. 727–774, 1994.

[15] B. Gruschko, D. Kolovos, and R. Paige, "Towards Synchronizing Models with Evolving Metamodels," in *Proc. 1st Int. Workshop on Model-Driven Software Evolution*, 2007.

[16] E. Guerra, J. de Lara, M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger, "Automated Verification of Model Transformations Based on Visual Contracts," *Automated Software Engineering*, vol. 20, no. 1, pp. 1–42, 2013.

[17] G. Guerrini and M. Mesiti, "X-Evolution: A Comprehensive Approach for XML Schema Evolution," in *Proc. of 19th Int. Conf. on Database and Expert Systems Application*, 2008.

[18] G. Guerrini, M. Mesiti, and D. Rossi, "Impact of XML Schema Evolution on Valid Documents," in *Proc. of 7th Int. Workshop on Web Information and Data Management*. ACM, 2005, pp. 39–44.

[19] M. Hartung, J. Terwilliger, and E. Rahm, "Recent Advances in Schema and Ontology Evolution," in *Schema Matching and Mapping*. Springer-Verlag, 2011.

[20] M. Herrmannsdoerfer, S. Benz, and E. Juergens, "Automatability of Coupled Evolution of Metamodels and Models in Practice," in *Proc. of 11th Int Conf. on Model Driven Engineering Languages and Systems*, 2008.

[21] ——, "COPE - Automating Coupled Evolution of Metamodels and Models," in *Proc. of 23rd European Conf. on Object-Oriented Programming*, 2009.

[22] L. Iovino, A. Pierantonio, and I. Malavolta, "On the Impact Significance of Metamodel Evolution in MDE," *Journal of Object Technology*, vol. 11, no. 3, pp. 1–33, 2012.

[23] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A Model Transformation Tool," *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31–39, 2008.

[24] F. Keienburg and A. Rausch, "Using xml/xmi for tool supported evolution of uml models," in *In Proc. of 34th nd Hawaii International Conference on System Sciences*, 2001, pp. 9064–.

[25] C. Krause, J. Dyck, and H. Giese, "Metamodel-Specific Coupled Evolution Based on Dynamically Typed Graph Transformations," in *Theory and Practice of Model Transformations*. Springer-Verlag, 2013.

[26] S. Kruse, "On the Use of Operators for the Co-Evolution of Metamodels and Transformations," in *Proc. of 5th Int. Workshop on Models and Evolution*, 2011.

[27] A. Kusel, J. Schönböck, M. Wimmer, G. Kappel, W. Retschitzegger, and W. Schwinger, "Reuse in Model-to-Model Transformation Languages: Are we there yet?" *Software and Systems Modeling*, pp. 1–31, 2013.

[28] R. Lämmel, "Coupled software transformations," in *Proc. of 1st Int. workshop on Software Evolution through Transformations*, 2004.

[29] R. Lämmel and W. Lohmann, "Format Evolution," in *Proc. of Re-Technologies for Information Systems*, 2001.

[30] F. Legler and F. Naumann, "A Classification of Schema Mappings and Analysis of Mapping Tools," in *Proc. of 12. Fachtagung in Datenbanksysteme in Business, Technologie und Web*, 2007.

[31] M. M. Lehman, "Laws of Software Evolution Revisited," in *Proc. of 5th Europ. Workshop on Software Process Technology*, 1996.

[32] ——, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[33] S. Lehnert, "A review of software change impact analysis," Technische Universtität Ilmenau, Tech. Rep., 2011.

[34] T. Levendovszky, B. Rumpe, B. Schätz, and J. Sprinkle, "Model evolution and management," in *Model-Based Engineering of Embedded Real-Time Systems*. Springer-Verlag, 2010.

[35] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, 2012, online Version.

[36] F. Mantz, G. Taentzer, and Y. Lamo, "Well-formed Model Co-evolution with Customizable Model Migration," *Electronic Communications of the EASST*, vol. 58, p. 14, 2013, online Version.

[37] A. Maule, W. Emmerich, and D. Rosenblum, "Impact Analysis of Database Schema Changes," in *Proc. of 30th Int. Conf. on Software Engineering*, W. Schäfer, M. B. Dwyer, and V. Gruhn, Eds. ACM, 2008, pp. 451–460.

[38] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, 2004.

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

[40] A. Narayanan, T. Levendovszky, D. Balasubramanian, and G. Karsai, "Automatic Domain Model Migration to Manage Metamodel Evolution," in *Model Driven Engineering Languages and Systems*. Springer-Verlag, 2009.

[41] N. F. Noy, A. Chugh, W. Liu, and M. A. Musen, "A Framework for Ontology Evolution in Collaborative Environments," in *Proc. of 5th Int. Conf. on The Semantic Web*, 2006.

[42] G. Papastefanatos, P. Vassiliadis, A. Simitsis, and Y. Vassiliou, "Hecataeus: Regulating schema evolution," in *Proc. of 26th IEEE Int. Conf. on Data Engineering*, 2010.

[43] M. Pizka and E. Jurgens, "Automating Language Evolution," in *Proc. of 1st Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*, 2007.

[44] L. M. Rose, M. Herrmannsdoerfer, J. R. Williams, D. S. Kolovos, K. Garcés, R. F. Paige, and F. A. Polack, "A Comparison of Model Migration Tools," in *Model Driven Engineering Languages and Systems*. Springer-Verlag, 2010.

[45] L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Model Migration with Epsilon Flock," in *Proc. of 3rd Int. Conf. on Theory and Practice of Model Transformations*, 2010.

[46] D. Ruscio, L. Iovino, and A. Pierantonio, "A Methodological Approach for the Coupled Evolution of Metamodels and ATL Transformations," in *Theory and Practice of Model Transformations*. Springer-Verlag, 2013.

[47] J. Sprinkle and G. Karsai, "A Domain-Specific Visual Language For Domain Model Evolution," *Journal of Visual Languages & Computing*, vol. 15, no. 3, pp. 291–307, 2004.

[48] J. Sprinkle, B. Rumpe, H. Vangheluwe, and G. Karsai, "Metamodelling: State of the Art and Research Challenges," in *Proc. of the 2007 Int. Dagstuhl Conf. on Model-based engineering of embedded real-time systems*, 2010.

[49] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic, "User-Driven Ontology Evolution Management," in *Proc. of 13th Int. Conf. on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, 2002.

[50] M.-A. Storey, "Theories, tools and research methods in program comprehension: past, present and future," *Software Quality Journal*, vol. 14, no. 3, pp. 187–208, 2006.

[51] H. Su, D. Kramer, L. Chen, K. Claypool, and E. A. Rundensteiner, "XEM: Managing the evolution of XML documents," in *Proc. of 11th Int. Workshop on Research Issues in Data Engineering*, 2001.

[52] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the Use of Higher-Order Model Transformations," in *Model Driven Architecture - Foundations and Applications*. Springer-Verlag, 2009.

[53] E. Visser, "A Survey of Strategies in Rule-Based Program Transformation Systems," *Journal of Symbolic Computation*, vol. 40, no. 1, pp. 831–873, 2005.

[54] G. Wachsmuth, "Metamodel Adaptation and Model Co-adaptation," in *Proc. of the 21st Europ. Conf. on Object-Oriented Programming*. Springer-Verlag, 2007.

[55] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger, "Towards an expressivity benchmark for mappings based on a systematic classification of heterogeneities," in *Proc. of 1st Int. Workshop on Model-Driven Interoperability*, 2010.

[56] M. Wimmer, A. Kusel, J. Schönböck, W. Retschitzegger, W. Schwinger, and G. Kappel, "On using Inplace Transformations for Model Co-Evolution," in *Proc. of 2nd Int. Workshop on Model Transformation with ATL*, vol. 10, 2010, pp. 65–78.

[57] M. Wimmer, G. Kappel, A. Kusel, W. Retschitzegger, J. Schönböck, and W. Schwinger, "Surviving the Heterogeneity Jungle with Composite Mapping Operators," in *Proc. of 3rd Int. Conf. on Theory and Practice of Model Transformation*, 2010.