Article:
Paige, Richard Freeman, Matragkas, Nikolaos and Rose, Louis Matthew (2016) Evolving models in Model-Driven Engineering: State-of-the-art and future challenges. Journal of Systems and Software. pp. 272-280. ISSN: 0164-1212

https://doi.org/10.1016/j.jss.2015.08.047

Evolving Models in Model-Driven Engineering: State-of-the-Art and Future Challenges

Richard F. Paige, Nicholas Matragkas and Louis M. Rose

Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, United Kingdom

Abstract

The artefacts used in Model-Driven Engineering (MDE) evolve as a matter of course: models are modified and updated as part of the engineering process; metamodels change as a result of domain analysis and standardisation efforts; and the operations applied to models change as engineering requirements change. MDE artefacts are inter-related, and simultaneously constrain each other, making evolution a challenge to manage. We discuss some of the key problems of evolution in MDE, summarise the key state-of-the-art, and look forward to new challenges in research in this area.

Keywords: evolution, migration, co-evolution, metamodel, transformation

1. Introduction

Software engineering – like many engineering disciplines – is all about managing constraints: on the systems we want to build, those that come from the development and business processes we operate, those from the organisational context in which we work, and those from the people who build the software. These constraints must be specified, related, and managed to ensure that we build software that satisfies its requirements and does not invalidate its contextual obligations. Different software engineering approaches and methodologies attempt to manage constraints in different ways. Some approaches treat constraints as mathematical entities, and develop rigorous theories for their manipulation and management (e.g., [13]). Others treat constraints informally and deploy software engineering practices (such as use of metaphors and pair programming) to manage them indirectly (e.g., [4]). Many approaches fit in between these two perspectives.

Model-Driven Engineering (MDE) [28] is a modern software engineering approach that attempts to present a unified conceptual model for how systems engineering should take place. Engineering builds and operates on precise and

Email address: [richard.paige, nicholas.matragkas, louis.rose]@york.ac.uk (Richard F. Paige, Nicholas Matragkas and Louis M. Rose)

structured *models* which are created for a particular purpose, and are themselves manipulated by software tools (e.g., for generating code) [56]. Software engineering processes therefore involve constructing, storing, modifying, analysing and destroying models. Substantial research has been carried out on MDE, and many excellent tools exist to support different MDE tasks (such as generating code from abstract models), e.g., [30, 41, 23, 35].

The models created when using MDE typically are defined in modelling languages, which are precisely specified and defined using *metamodels* [42]. Metamodels are, informally, a set of constraints that distinguish well-formed from ill-formed models: a valid model is said to *conform* to its metamodel. A metamodel describes the abstract syntax, and certain static semantic properties (such as multiplicities of elements involved in relationships), of modelling languages. This is, at least superficially, no different from other language-oriented approaches to software engineering, such as formal methods, where a precisely defined language is used to specify artefacts, and those artifacts are manipulated over the course of development. However, MDE has a number of significant differences:

- *Tools come first in MDE:* the modelling languages that are used are designed so as to be supported by tools – e.g., editors, syntax highlighters, debuggers, etc. Standard frameworks, such as EMF [57], exist to help define modelling languages in such a way so as to support this. In contrast, formal specification languages are designed to support mathematical reasoning, and as such the priority is to have a sound and complete mathematical semantics, which thereafter be supported by tools.

- *Automated processing of models:* models are meant to be processed by tools. This is the discipline of *model management* [53, 43]. Typical model management tasks include transformation, comparison, merging, migration, validation and text generation, though the specific tasks that are used in industrial application are driven and dictated by engineering process and organisational context (e.g., use of automated code generation from models that enables audit and inspection).

- *Languages are themselves models:* a metamodel is itself a model, and can be instantiated; in some interpretations of MDE, the operations on models (e.g., transformations) are also models with their own metamodels. Metamodels are also instances of so-called metametamodels, thus (at least conceptually) making it easier to build generic tools that support a wide variety of languages and modelling styles.

Models provide a unified conceptual way of thinking about and carrying out software engineering, and the tools that exist can help to make certain aspects of MDE work in practice [6]. But use of models is not without significant challenges. Software engineering must always be prepared to deal with change: of requirements, of technology platforms, of developers, etc. In some cases, change and its impact on software engineering may be difficult to identify or

assess. Both challenges and opportunities arise with managing change in MDE. One significant opportunity comes from MDE making *dependencies* between artefacts (models, metamodels, operations on models) – typically in the form of constraints – explicit: these relationships can, in principle, be used to identify and calculate the impact of changes (to models, metamodels, operations) on other artefacts. A challenge arises from the significant number of constraints in place in MDE: models, metamodels and operations thereupon are effectively sets of constraints on what can be specified, and what can be done to those specifications. MDE can be reduced, conceptually, to a constraint management problem: trying to building satisfiable sets of constraints whose instances are solutions to important problems.

A significant issue associated with constraint management in any discipline is dealing with *change*: change in the structures to which constraints are applied; changes to individual constraints; changes to sets of constraints; or changes to the tools that evaluate constraints on structures. Constraints invariably depend on each other, so making even small changes can have significant impact on other, related artefacts. Change management for MDE is a particular instance of this problem, and there are many specialisations within MDE. One that has seen particular recent interest is *model evolution*: models changing over time, typically in response to some kind of external event. Numerous approaches have been developed to help systematise the process of model evolution. In this paper, we highlight the state-of-the-art in model evolution, and relate it to future challenges in research in this area. But first, we briefly contextualise the discussion by providing a short overview of evolution in MDE in general, before turning to our survey.

2. Context: Evolution in MDE

When building systems using MDE, the trinity of artefacts that is used is:

- *Metamodels*, which describe the structures and rules applicable for a family of models (metamodels themselves are models, which conform to a metametamodel).
- *Models*, which are particular instances of structures and rules.
- *Operations*, which are defined on metamodels and apply to models (e.g., transformation, comparison, merging). Operations may have side-effects: particuarly, they may generate *trace-links* which relate model elements (e.g., the source of a transformation to the target).

When a model uses the structures and obeys the rules of a metamodel, it is said to conform to the metamodel. Formally, conformance can be defined as a set of constraints that hold (at precisely defined times) between a model and a metamodel.

MDE involves constructing models (e.g., of requirements, architectures, designs, code, tests) and applying operations to said models in order to automate

parts of the engineering process. Operations encapsulate some of the engineering know-how and decision making that is inherent in the design process. There are several critical observations pertaining to this:

- Artefacts are *formal entities*, in the sense that precise (and often standardised) specifications of what constitutes a valid metamodel, model and operation exists.
- All artefacts are inter-related (models conform to metamodels, metamodels themselves conform to metametamodels, operations are defined in terms of metamodels and apply to models), and these inter-relationships are formal (in the sense that precise specifications exist, e.g., of model conformance).
- MDE processes are formal, in the sense that formally defined operations can be used to implement significant parts of them.
- Heterogeneity is inherent with these artefacts. MDE typically uses different languages (metamodels), different models, different types of operation (e.g., model-to-model transformations, model-to-text transformations), and even different model persistence technologies.
- Not all types of change are created equal. Gruschko et al [24] classified the types of metamodel change that could occur, and identified some changes that could not be processed automatically. This suggests that for change management in MDE, engineering judgement will always be needed, either to choose the most suitable approach for the problem at hand, or to choose from among a set of potential change management solutions.

Of course, all of these artefacts may need to change, and some changes may be more difficult to manage than others. Changes to models may inherently be part of the engineering process, and may be carried out by an engineer applying operations directly (e.g., an update-in-place transformation may be used to change a model). Changes to operations may be supported by an MDE expert responsible for writing and maintaining the operation. Changes to metamodels may need to be analysed and carried out by a language engineer. Any change to an artefact may result in changes to other artefacts. This is particularly the case for changes to metamodels, which may require changes to operations that use those metamodels, and changes to all models that conform to the previous version of the metamodel. Changing metamodels is a process that is related to changing programming language APIs (both conceptually and pragmatically), though arguably changing metamodels has significant differences, in part because metamodels can capture more than just static programming interface details.

Arguably, MDE has all of the challenges of evolution and change management inherent in other software engineering disciplines (e.g., code-centric or data-centric approaches, where language feature obsolescence, versioning, change propagation and change management are all issues) – plus new ones:

the languages used in MDE are themselves software artefacts and are amenable to change; and the dependencies between artefacts, while many, are made explicit and as such applicable to evolution and change management processes.

2.1. Key characteristics of evolution solutions in MDE

MDE approaches to managing evolution can largely be divided into two groups: those where metamodels don't change, and those where they do. The former approaches make systematic use of model transformations; the latter typically involve abstractions over transformations, e.g., higher-order transformations that generate mappings, or operators that support migration scenarios. When attempting to manage evolution and change in MDE, there are a number of important characteristics associated with existing solutions.

- *Scope:* is the solution applicable to managing change for one type of MDE artefact (e.g., models) at a time, or more than one artefact at at time? Many of the current approaches for managing evolution in MDE are so-called *coupled* approaches, where changes to metamodels trigger changes to other MDE artefacts.

- *Automation:* to what degree is the solution automated? Current solutions range from manual approaches (which provide task-specific syntax and tools for building custom, flexible evolution solutions) to fully automated ones.

- *Environment:* to what degree is a specialised environment required in order to support MDE evolution? In particular, can standard modelling tools (e.g., Eclipse EMF [8]) be used to construct models, or must specialised model editors or moperation recorders be used?

- *Conformance:* when metamodel change is involved, evolution solutions must provide means for establishing and re-establishing conformance to a metamodel. For example, a metamodel might change, and models thereafter must be updated to conform to the new version of the metamodel. *When must conformance hold between model and metamodel?* In all approaches, conformance is an outcome of evolution/migration, but some approaches impose conformance at intermediate stages of the process, too (e.g., to enable richer forms of analysis and reasoning). Additional, treatment of constraints in conformance varies from approach to approach: in some, constraint checking is treated separately to metamodel conformance (e.g., in AML [20], Flock [49]), whereas in others, a unified treatment of conformance and constraints is provided.

As we will see, the state-of-the-art solutions for change management in MDE vary in their theoretical and practical approaches for addressing these characteristics.

3. State-of-the-art

In this section we describe the state-of-the-art in managing evolution in MDE. We consider solutions in a number of categories, and consider the key research findings and technological results.

3.1. Co-evolution of model and metamodel

Model-metamodel co-evolution solutions apply to two MDE artefacts at once – the intent is to update models so that they conform to an evolved metamodel. A variety of solutions have been proposed for model-metamodel co-evolution that vary in terms of their degree of automation, the environment required to carry out co-evolution, and when the conformance relationship between model and metamodel must hold.

Broadly speaking, there are three categories of co-evolution approaches in MDE: inference approaches, operator approaches, and manual approaches. Manual approaches are programmatic: engineers specify strategies, typically by hand, which migrate models to an updated metamodel. Inference approaches are based on so-called *comparison* or *differencing*: original and updated metamodels are compared, and the changes that have been identified are used to automatically or semi-automatically generate evolutionary strategies for models. Operator approaches are pattern-based, and encode a set of predetermined micro-strategies that, step-by-step, will allow (1) a metamodel to be evolved in systematic and predictable ways; and (2) models to be evolved to conform to the new metamodel, in systematic and repeatable ways.

All such solutions are partly automated – some metamodel changes (breaking and unresolvable changes) can only be processed manually, because they require domain expertise to resolve. Approaches vary in terms of how they deal with breaking and unresolvable changes: most approaches make it possible for engineers to intervene in an evolutionary process, while others rule such changes out of scope.

A well-known example of a *manual* approach to co-evolution is Ecore2Ecore [26], an EMF-specific tool that augments the EMF model loading facilities with programmatic migration strategies. The idea with Ecore2Ecore is that interrelationships between original and evolved metamodels are specified (e.g., Element is equivalent to NamedElement) and the tool automatically generates a partial migration strategy. The approach is quite limited, in that it does not support certain more complicated types of evolution (e.g., splitting of metaclasses, changing the types of properties), which must be processed manually using Java. Ecore2Ecore provides no special support for conformance checking: EMF checks conformance of the migrated model upon completion of processing; constraints may additionally be checked using other suitable tools (e.g., Eclipse OCL [60], EVL [32]).

A more abstract and automated approach is Epsilon Flock [50], which supports a notion of *conservative copy*, wherein parts of the original model that

remain conformant to the target metamodel are automatically copied[1]; by contrast Ecore2Ecore must programmatically copy *all* model elements. As such, Flock migration strategies tend to be much more concise than other manual approaches – and even other non-manual approaches to migration [48]. Conformance of model with metamodel applies when the migration strategy has completed its execution. What is telling about both Ecore2Ecore and Flock is that these approaches ignore *constraints* on models – that is, conforming models are defined entirely in terms of model-metamodel constraints, and do not take into account additional constraints that may apply to the metamodel (e.g., in OCL or EVL), which are checked separately from the migration/evolution process.

A manual approach that does consider additional constraints is presented in Taentzer et al [59], which defines a multi-stage evolution process. This approach is stricter in terms of enforcing conformance to the metamodel – co-evolution rules are specified using graph transformations, and conformance to a metamodel (and satisfaction of constraints applied to models) is guaranteed by construction after every migration step, i.e., after application of a set of graph transformations. This is useful particularly for formal reasoning about the validity of migration strategies.

Operator approaches – such as COPE/Edapt [25] or MCL [38] – are based on a pre-defined set of micro-strategies: patterns that map original to evolved metamodel elements. The premise with such approaches is that a set of such operators, when applied to a metamodel, will produce an evolved metamodel, and then higher-order transformations can be applied to generate a migration strategy for models that conform to the original metamodel. Such approaches are usually extensible, so that new operators can be designed and specified to deal with evolutionary scenarios not originally anticipated. However, these approaches normally require use of a specialised editor for constructing metamodels; these editors record the changes that are made as a metamodel evolves (in terms of the operators used to modify the metamodel), so as to make the automatic generation of migration strategies as tractable as possible. A related operator-like approach that makes use of constraints instead of micro-strategies is in [12]. It uses constraints to identify metamodel co-evolution *failures*, and automatically generates *repairs* to resolve such failures; this promising work, which is based on existing automatic inconsistency detection, is still at the prototype/proof-of-concept stage, unlike many of the other frameworks and tools described in this section. It also appears to be immediately applicable to co-evolution for UML [52], and hence relies on a specialised editing environment, so its applicability to other languages remains unclear. All such approaches check conformance of models to metamodels after the migration process has completed.

Finally, *inference* approaches are based on analysis and comparison of orig-

[1]Conservative copy is related to notions of frames in programming methodology and artificial intelligence.

inal and evolved metamodels. Such approaches – e.g., Cicchetti's [10] or AML [21] – make use of metamodel matching algorithms (such as similarity flooding [36]) to identify differences between metamodels, and use this to infer a record of changes that can be used to automatically generate a migration strategy. Similar to operator approaches, a higher-order transformation is used (e.g., in ATL [27]) to generate the migration strategy. Inference approaches are incomplete: the same change to a metamodel can, in general, be produced in a number of different ways, and inference approaches need engineer input to disambiguate such cases. However, inference approaches do not require use of a specialised editor/environment, and conformance of migrated model to evolved metamodel is checked on completion of the migration process.

A hybrid approach – which combines elements of inference and operators – that eliminates the need for editor-based tracking of operator application is [34]. It also makes a novel contribution in supporting composite operator detection based on difference models, thus addressing concerns related to granularity and scalability that apply to existing operator and inference approaches.

What are the key observations related to the state of the art in model-metamodel co-evolution? There is a significant tradeoff between automation and flexibility: those approaches that provide greater automation for co-evolution (e.g., COPE (now called Edapt[2]), Cicchetti's work [10]) give less control over the process. Another observation is that some of the co-evolution platforms (e.g., Edapt, AML, MCL) are restricted to a specific platform, either for modelling and metamodelling (EMF) or for implementing the generation of migration strategies (e.g., specific model-to-model transformation languages like ATL).

What is also noteworthy is that almost all approaches are based on carrying out model migration *after* metamodel evolution, where the latter triggers the former process: there is little work on attempting to evolve metamodels based on inference from examples of original and evolved *models*. Williams [61] explores the use of optimisation-based techniques for inferring model migration strategies from example models, but notes significant concerns about efficiency and performance. It seems likely that such approaches are much less tractable than alternatives, in part because they will need to apply search-based and optimisation techniques, which can be challenging to scale to large problems.

3.2. Co-evolution of operation and metamodel

Models will change frequently in many MDE processes, and metamodels may change frequently in the early (information gathering) stages of an MDE process that is based on development and use of domain-specific languages. The operations that are defined on metamodels, and apply to models, may also evolve in response to changes in requirements or changes in metamodels. So, what approaches exist to help support co-evolution of MDE operations (like transformations, comparisons, mergings) and metamodels? Approaches to co-evolution of operations and metamodels generally follow similar approaches to metamodel

[2] `http://www.eclipse.org/edapt/`

co-evolution: the metamodel is changed, and in some approaches a change model (which records the specific modifications that are made) is produced for later use. Secondly, the extent to which dependent artefacts (operations) have been affected by the metamodel evolution is assessed. Finally, change propagation is used to migrate the dependent artefacts.

The previous section discussed key approaches to performing change management for models and metamodels (e.g., inference, operators). There are significantly fewer approaches for metamodel-operation co-evolution (the problem is significantly harder, because it requires evolution and migration of operation semantics as well as structure), and all of this work seems to focus on model transformations, possibly because of the perceived importance of model transformations in MDE. Méndez et al [37] use a metamodel monitor to report changes, and process change events to select and execute an appropriate migration strategy for evolving model transformations; the approach is not fully automated, and engineer guidance is sometimes required to select a strategy. More generally, this approach could also be used for metamodel co-evolution; even more generally, such approaches can generally be classifed as *event-driven*, and event-driven operations have significant potential in MDE for addressing scalability concerns. Another key paper in this space is [54], which presents a methodology for coupled evolution of metamodels and transformations; what is particularly interesting about this approach is it attempts to assess the *cost* of a change and uses this information to inform the evolution of ATL transformations (e.g., to make a "go/no-go" decision on whether to evolve a transformation).

Roser et al [51] focus on the evolution of transformations that are generated from the bindings between an ontology and a source or target metamodel. Metamodel evolution is restricted to cases where the original and the evolved elements remain bound to the same ontological concept, and, transformations are automatically migrated by substituting the original meta-element with the evolved meta-element. Such an approach may also be applicable for evolution of operations applied to domain-specific languages, where metamodel concepts are bound to, or derived from, a domain model.

3.3. Other approaches

While substantial research has focused on model-metamodel co-evolution, and a very small amount has considered operation-metamodel co-evolution, there have been approaches that sit outside of these broad categories of research.

3.3.1. Round-trip engineering

One of the most fundamental approaches to managing evolution in MDE is *round-trip engineering* (sometimes called synchronisation): model transformations are used (e.g., to produce executable code from models), and after code has been modified, updated models are regenerated from code. Such approaches are specific for model evolution, and generally assume that metamodels remain unchanged. Naive approaches tend to struggle with non-trivial code bases and

large models; more nuanced approaches make use of *deltas*, or change models, to only make relevant changes to source models while leaving unaffected parts unchanged. Of significant note in this body of work is the FUJABA toolset, which makes use of triple graph grammars to support incremental synchronisation [22]. Incremental approaches to round-trip engineering have been highly influential in research on change propagation in model-metamodel co-evolution.

3.3.2. Model evolution as a transformation problem

An important category of research has taken the view that model evolution is a *transformation* problem, and as such can be directly addressed using model transformation languages. This is an appealing perspective, and there seems to be evidence to support this view, given the key role that transformations take in many of the solutions described in the previous section. The Henshin toolset [33] has considered use of graph transformations to support model migration and evolution. What is novel about this work is that the graph transformations (which carry out the model migration) are automatically generated from metamodel evolution mappings (e.g., that relate original and target meta-elements). Rose [46] has also considered the use of model transformations to support model migration, but observes that there are patterns inherent in the transformations (e.g., the so-called conservative copy pattern) that are not directly supported by transformation languages, and these lead to transformation blow-up. Such issues are addressed in Flock [50] by embedding some evolutionary patterns as syntactic constructs, and others (e.g., conservative copy) in the semantics of the language, whereas in Henshin, higher-order generative techniques are applied.

Both Henshin and Flock are general-purpose migration tools; a problem-specific tool is GMF Evolution[3], which is specifically for evolving GMF models when an EMF model changes [55].

3.3.3. Schema evolution

Much of the work on model evolution has been inspired by, or borrowed directly from the wealth of research on schema evolution in databases [45, 44]. While many of the key ideas from schema evolution have been directly adopted or adapted for model evolution, one key consideration has yet to be fully addressed: after schema evolution and data migration, there is also a need for application evolution, so that applications can be refactored to be compatible with the new schema. While MDE research on operation-metamodel co-evolution is one step along these lines, more research needs to be carried out, particularly to explore how applications developed without use of MDE, but that interface with metamodels, can be migrated automatically and efficiently.

3.3.4. Model Versioning

Model versioning plays an important role in managing evolution in the context of MDE. Following [17] the goal of versioning in software engineering is

[3] `http://www.emfmigrate.org/gmf-evolution/`

twofold. First, versioning is concerned with maintaining a historical archive of a set of artefacts (models and metamodels in the case of MDE), as they undergo a series of changes. Second, versioning supports the simultaneous evolution of an artefact by a team of engineers. As such, model versioning may be applicable and useful in collaborative MDE scenarios involving model migration (e.g., when two or more versions of a metamodel need to be simultaneously supported in a project).

Versioning of metamodels and models is quite different from versioning textual artefacts such a source code. This is due to the graph-based structure of modeling artefacts. Text-based versioning systems such as Subversion[4] or Git[5] consider only the textual information of a modeling artefact such as its XMI serialisation. In doing so they fail to take into consideration the structural information of models such as containment references or the multiplicities between different model elements. Therefore, graph-based versioning techniques and systems are needed.

In a typical model versioning scenario, parallel modifications on a common version V_0 of a modelling artefact result to a set of modified versions $\mathcal{V}=\{V_1, V_2, \ldots, V_n\}$, where n is the number of engineers working in parallel on that artefact. The goal of the model versioning system is to consolidate the modified versions in $\mathcal{V}$ into a new version $V_0{}'$.

A model versioning process consists of three distinct phases, namely change detection, conflict detection and inconsistency detection [1]. In the change detection phase, the set of changes between the common version V_0 and the subsequent versions in $\mathcal{V}$ are identified. Change detection can be realised using either state-based approaches or operation-based approaches. State based approaches identify model changes by considering only the final states of the modified versions, i.e. by considering only the versions is $\mathcal{V}$. On the other hand, operation-based approaches rely on a model editor to keep track of all the operation sequences applied to the original version V_0 and lead to the final versions in $\mathcal{V}$. Although, state-based approaches support only atomic operations such as deletions, additions, and updates, operation-based approaches can additionally support composite operations such as model refactorings. However, the additional support provided by operation-based approaches comes at a price. Such approaches have a strong dependency on a modelling editor and they are usually language-specific.

The detection of changes in the change detection phase sets the basis for the two subsequent phases, namely conflict and inconsistency detection. Conflicts occur when overlapping and contradicting modifications occur concurrently by different engineers. Such conflicts can be detected by comparing all the changes incurred by a model element and finding which ones are overlapping and contradicting. Conflicts can be resolved either manually or automatically. Manual conflict resolution for modelling artefacts can be challenging. In the case of

[4] http://subversion.apache.org/
[5] http://git-scm.com/

textual artefacts two conflicting versions are shown to the user side by side and the conflicting areas are highlighted. Then the user has to take action in order to reconcile the two versions. In the case of modelling artefacts, this approach is not straightforward. Models in MDE have dual representations manifested by their abstract and concrete syntax. Consolidating these representations separately can be a challenging task [7].

On the other hand, automatic conflict resolution can be achieved by calculating all possible combinations of parallel performed operations leading to a valid version. Cicchetti et al. [9] propose a domain-specific language for defining conflict patterns and reconciliation strategies. These patterns and strategies can be used to detect and reconcile both syntactical and semantic conflicts. Policy based approaches like the one proposed by [9], require user intervention in scenarios where no policy is specified. To address this issue [16] propose a formalised conflict resolution strategy for conflicts based on graph modifications which results in a consolidated model by construction. Finally, instead of trying to resolve conflicts a model versioning system can just tolerate them. Researches argue that temporarily tolerating conflicts can be beneficial since these conflicts highlight areas of the system where further analysis is needed [39].

Finally, the inconsistency detection phase takes place after the modified versions have been merged to a new, consolidated version. During these phase consistency problems introduced by the merging activity are detected. Such problems usually occur when the merged model violates any validation rules associated with the metamodel. Such inconsistencies are usually resolved manually.

Several model versioning systems have been proposed in the literature. These systems employ different combinations of the aforementioned techniques to support model versioning. One of the earliest works on versioning UML models was published in [58], which present metamodel-independent algorithms for differencing, merging, and conflict resolution. Their approach calculates differences between versions of the same model by matching the unique identifiers of model elements and by calculating the created, deleted and changed elements. One main limitation of their work is that they do not consider composite operations.

EMFStore is a model repository for EMF models proposed in [29] and it provides dedicated facilities for model versioning. This tool is operation-based and it tracks all the modifications undergone by a model, once it is checked out. These modifications are then committed to the repository and the latest version is updated. The tool supports also transactions, i.e. groups of atomic operations which are treated as a single composite operation. If a conflict occurs in one of the atomic operations of a transaction, then the entire transaction is marked as conflicting and resolution is required.

Odyssey-VCS 2 is a version control system dedicated to UML models [40]. The tools uses state-based differencing which relies on unique identifiers to detect corresponding elements between different versions of a model. Based on the matching phase, the tool infers the atomic operations, which led to the latest version of the model. Composite operations are not considered. If a model element is modified in two different ways at the same time, then a conflict warning

is raised. Finally, the tool does not apply any inconsistency detection after the different versions are merged.

Finally, *AMOR* (Adaptable Model Versioning) is a model version control system presented in [2]. AMOR provides a conflict detection mechanism, which can be extended by the user with custom composite operations. Moreover, AMOR provides a recommender components, which provides suggestions to users on how to resolve detected conflicts. Finally, the tool supports the definition of conflict resolution policies in a collaborative manner.

4. Challenges

There has been substantial research in models and evolution over the past ten years, but numerous challenges remain; some are technical and theoretical, others are more focused on engineering practice and process. We identify some of the key ongoing challenges in this section.

4.1. Scalability

A fundamental challenge for MDE – and not just evolution in MDE – is scalability. MDE theories and tools need to support large metamodels (like UML 2.x, AUTOSAR [19], EAST-ADL [11]), small metamodels (like those artisanal languages created as one-offs), and everything in between. They need to support the construction of large and small models, as well as operations that execute on large models. Support for evolving very large models is required. This may mean providing facilities for *ignoring* parts of the model that are out of scope for solving migration and evolution problems, for better decomposition of metamodels (e.g., into parts that are unlikely to evolve and parts that may evolve), for more scalable persistence mechanisms for models, and new architectures for model management operations (e.g., transformations that react to triggered events, such as changes to parts of a model). Many of these concepts are well-understood in fields other than MDE, but translational research will be needed to exploit them here. Some of these issues are being studied in the context of the EU FP7 MONDO project[6].

4.2. Managing automation

One of the leading principles of MDE is *automate repetitive and error-prone tasks*; operations like transformations, comparisons, validations and mergings aim to encapsulate some of the automation patterns that have been identified for MDE. There is always a trade-off between automation and control: once a task has been (partly or completely) automated, the degree of fine control over how the task operates, and what it produces, is likely reduced. In MDE, automation arguably makes supporting evolution more difficult. In a model management workflow, each operation, metamodel and model may evolve, and this evolution must be managed systematically and precisely. Operations may generate

[6] `http://www.mondo-project.org/`

traceability information as a side-effect, and this information may evolve as well (for example, trace-links generated as a side-effect by a model transformation may themselves be models, conforming to a traceability metamodel, which may evolve to address new traceability scenarios). The particular challenge here may be the close coupling between *operation* and *metamodel* – we ideally want to automate as much of an engineering process as possible, but the automation (in the form of operations) depends on metamodels, which may be subject to change. At least two interesting research directions are available here: exploring *genericity* in model management, to reduce the coupling between operations and metamodels; and *flexible* model management, which attempts to reduce the role played by metamodels in MDE (particularly in early lifecycle stages of language engineering). This is discussed more shortly.

4.3. Dependency heterogeneity

The three key artefacts of MDE – models, metamodels and operations – are all inter-dependent, and much of MDE evolution requires managing and controlling these inter-dependencies. A significant challenge with this is managing the *heterogeneity* of inter-dependencies: many different kinds of relationships exist between MDE artefacts. For example, between model and metamodel there is a *conformance* relationship. Between model and operation there are (at least) *parameter* and *generates* relationships. A sound, precise theory of heterogeneous dependencies between MDE artefacts is needed, as well as compliant and pragmatic tool support. A promising direction for the former is *delta lenses* [18, 15], which is categorical in nature, but also has connections to research on traceability in MDE, and may also have an impact on research related to megamodelling [14].

4.4. Empirical studies

Gruschko et al [24] identified different kinds of metamodel change and classified those changes in terms of how resolvable they were via automated means. This work has proven influential in terms of setting requirements for tools to support model migration and model-metamodel co-evolution: all tools are generally compared against Gruschko's classification and aim to address all but the so-called *breaking, non-resolvable* changes. However, a more nuanced and empirical classification is yet to be produced. In particular, we need answers to the question: *which types of metamodel changes are most frequent in deployed metamodels?* With such information, we can target our model migration and co-evolution solutions to those most commonly occurring evolution problems, and gain increased confidence that our solutions are actually addressing the most important MDE evolution problems.

4.5. Usability of tools

Much of the research on models and evolution so far has focused on theories and end-user tools: languages for specifying model migration rules, generators

for model migration strategies, differencing algorithms, mathematical frameworks for expressing change operators, etc. The usability of these tools for solving end-user problems has not been explored thoroughly and systematically. Many of the tools that exist (e.g., Flock, Henshin [3], Edapt) are Eclipse-based and their usability is closely related to the usability (or lack thereof) of Eclipse. Additionally, syntactic and semantic compatibility with other model management operations has also informed the development of model migration and co-evolution solutions: builders of migration/co-evolution tools want their solutions to be familiar and easy to learn for users of other Eclipse-based model management tools. It remains to be seen whether these constraints lead to usable tools for end-users solving model migration problems; a thorough user study, as well as consideration of user requirements that are not directly taken or influenced by Eclipse, would be a worthwhile direction for future research.

4.6. Concrete syntax

Metamodel evolution is often the trigger for MDE migration problems: migrating models or operations to the new metamodel. A topic that has not received nearly as much attention as other aspects of MDE evolution is dealing with *concrete syntax*: how to evolve concrete syntax as a result of metamodel evolution or model migration? Many approaches to concrete syntax in MDE are generative (e.g., EuGENia) based on the metamodel, so one plausible solution for concrete syntax evolution would be to update the generators and regenerate any editors. However, for large metamodels, or complex languages, this may be inefficient. It may also be challenging in cases where several concrete syntaxes depend on the same abstract syntax/metamodel. Thus, we see a case for more significant research in better support for concrete syntax, e.g., via visitor-based approaches.

4.7. Run-time

Language evolution can happen at any time, e.g., upon issuance of a new version of a standard, as a result of changes to tool infrastructure (e.g., EMF). An assumption for most research on models and evolution is that migration of models and operations happens "off-line", i.e., when model editors are not being used, when transformations are not being executed, etc. It is not clear that this assumption will hold for large-scale MDE architectures – e.g., those on the cloud, those for handling streaming transformations of continuous models – which may be needed to support very large languages and very large models. It may be that evolutionary processes and migration needs to be carried out *while* a language is being used by engineers, e.g., the form of a model editor, language workbench, model transformation tool. To support such run-time adaptation, we will likely need further research in new ways of decomposing language definitions, and decoupling operations from language definitions, along the lines of the generic approaches used in generative software development [47].

4.8. Hybrid migration approaches

As discussed earlier, there are a number of migration approaches available, and these are largely classified as operator-based, inference-based or manual. It may be possible to achieve significant benefits in performance and scope of applicability by combining different types of techniques to form hybrid migration solutions. For example, a solution where a manual migration strategy is used for all but breaking, non-resolvable changes could be supplemented with a set of Edapt-like operators from which an engineer can choose. Inference-based approaches could also be used in concert with operator-based approaches to reduce the search space to which inference algorithms are to be applied.

4.9. External interfaces

MDE processes and tools do not and should not operate in a vacuum: they must interface with other non-MDE processes (e.g., for quality assurance, certification, early requirements elicitation) and tools (e.g., compilers, debuggers, continuous integration servers), not to mention organisational policies and processes. Metamodel evolution, model migration and operation evolution may have an impact on interoperation and integration with non-MDE processes and tools: applications may exist that interface with or operate on metamodels (e.g., via an API), and these must also be migrated to operate on new versions of metamodels. An interesting example of this type of problem is presented in [5]. A related issue is the relationship between evolution in MDE and evolution of code; if we are working in an environment where both MDE and code-based artefacts are evolving, how do we integrate the separate evolutionary processes? More generally, we argue that evolution of MDE artefacts must be part of the enterprise architecture of a (mature) organisation that exploits MDE. In the specific, we may be guilty of trying to automate change management of artefacts that are cheaper to throw away and rebuild, or that have a very limited lifespan within an organisation. Considering MDE evolution within the organisation's systems, processes, policies and rules should help to make such decisions (or, at least, will help to make the trade-offs explicit).

4.10. Language semantics

The focus of much MDE research has been on abstract syntax (metamodels) and concrete syntax (editors, graphical frameworks such as GMF and Graphiti); semantics is relatively undertreated by comparison. There are open issues related to maintenance of semantics via model migration. If a model is migrated from language X to language Y, is it possible – via the migration process – to provide guarantees of either semantics preservation of features from language X when encoded in Y, or preservation of other desirable properties? Some consideration of semantics preservation has been carried out, e.g., in work on COPE [25], though both the theory and practice of semantics preservation under model migration is still underdeveloped.

4.11. Metamodels

Models depend on metamodels; MDE operations depend on metamodels. Managing evolution for models requires managing the evolution of metamodels. Most solutions to model evolution and co-evolution have focused on metamodels. A different approach would be to *discard* metamodels entirely – take the view that they get in the way of efficiently supporting evolutionary processes. To what extent can we support MDE *without* metamodels, e.g., by using schema-less XML, or drawing tools such as yEd [31]? Can we use flexible modelling tools (such as GenMyModel[7] or EuGENia Live[8]) to support rigorous MDE processes?

Metamodels do provide significant benefits: they effectively provide a type system for models that allow us to verify and validate properties. But evolution and migration involving metamodels can be very expensive. Another interesting avenue of investigation may be to explore *when* in an MDE process metamodels can most effectively be imposed. For example, an MDE process may start with using flexible drawing tools like yEd, and then a metamodel is imposed once suitable domain analysis has been carried out (and the *implicit* metamodel behind the models has stabilised). This (early-phase) metamodel-less MDE might have some of the same tradeoffs as the schema-less data stores popularised by the NoSQL movement.

Metamodels are important and useful, but we should not be dogmatic in terms of *when* they should be applied in MDE.

5. Conclusions

The key principle of MDE is automating repetitive and error prone tasks. The decisions that we make, with respect to use of particular technologies and theories, the implementation of particular tasks, and the deployment of workbenches to users, should always aim to support that principle. The techniques and tools that have been developed to support evolution and migration in MDE all address the flexibility-automation tradeoff (like other MDE operations, such as transformation), but in doing so reveal a great deal about how constrained – and arguably overconstrained – MDE really is. Managing evolution in MDE resolves to managing sets of inter-related constraints between artefacts. As such, if we want to maximise automation, we are significantly restricted in terms of how much control we have over the process, and still cannot automate everything. Perhaps one of the problems is the dependence on metamodels. Perhaps we should see how far we can go with using the good ideas of MDE – such as automating tasks, building domain-specific languages and working with domain experts (i.e., software language engineering) – without assuming that metamodels are required. Metamodels add value, but *when* in the engineering process (assuming that evolution will happen) do they maximise their value?

[7] `http://www.genmymodel.com/`
[8] `http://eugenialive.herokuapp.com`

Acknowledgements. The authors thank the editors for their feedback and for the invitation to write this article.

References

[1] Kerstin Altmanninger, Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. Why Model Versioning Research is Needed!? An Experience Report. In *Proceedings of the Joint MoDSE-MC02CM 2009 Workshop*, 2009.

[2] Kerstin Altmanninger, Gerti Kappel, Angelika Kusel, Werner Retschitzegger, Martina Seidl, Wieland Schwinger, and Manuel Wimmer. AMOR - Towards Adaptable Model Versioning. In *1st International Workshop on Model Co-Evolution and Consistency Management (MCCM'08), Workshop at MODELS'08*, Toulouse, France, 2008.

[3] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: advanced concepts and tools for in-place emf model transformations. In *Model Driven Engineering Languages and Systems*, pages 121–135. Springer, 2010.

[4] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, October 1999.

[5] Mathieu Beine, Nicolas Hames, Jens H. Weber, and Anthony Cleve. Bidirectional transformations in database evolution: A case study "at scale". In *EDBT/ICDT Workshops*, pages 100–107, 2014.

[6] Jean Bézivin. On the unification power of models. *Software and System Modeling*, 4(2):171–188, 2005.

[7] Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. An introduction to model versioning. In *Proceedings of the 12th International Conference on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-driven Engineering*, SFM'12, pages 336–398, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] Frank Budinsky and Stephen A Brodsky. Merks, eclipse modeling framework, 2003.

[9] Antonio Cicchetti, Davide Di Ruscio, and Alfonso Pierantonio. Managing model conflicts in distributed development. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus V02lter, editors, *MoDELS*, volume 5301 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2008.

[10] Antonio Cicchetti, Davide Di Ruscio, and Alfonso Pierantonio. Managing dependent changes in coupled evolution. In *ICMT*, pages 35–51, 2009.

[11] Vincent Debruyne, Françoise Simonot-Lion, and Yvon Trinquet. Eastadlan architecture description language. In *Architecture Description Languages*, pages 181–195. Springer, 2005.

[12] Andreas Demuth, Roberto E. Lopez-Herrejon, and Alexander Egyed. Coevolution of metamodels and models through consistent change propagation. In *ME@MoDELS*, pages 14–21, 2013.

[13] Antoni Diller. *Z: An introduction to formal methods*, volume 2. Wiley England, 1990.

[14] Zinovy Diskin, Sahar Kokaly, and Tom Maibaum. Mapping-aware megamodeling: Design patterns and laws. In *SLE*, pages 322–343, 2013.

[15] Zinovy Diskin and T. S. E. Maibaum. Category theory and model-driven engineering: From formal semantics to design patterns and beyond. In *ACCAT*, pages 1–21, 2012.

[16] Hartmut Ehrig, Claudia Ermel, and Gabriele Taentzer. A formal resolution strategy for operation-based conflicts in model versioning using graph modifications. In Dimitra Giannakopoulou and Fernando Orejas, editors, *Fundamental Approaches to Software Engineering*, volume 6603 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2011.

[17] Jacky Estublier, David Leblang, André van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, and Darcy Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430, October 2005.

[18] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3), 2007.

[19] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkelin, Kenji Nishikawa, and Klaus Lange. Autosar–a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, 2009.

[20] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Managing Model Adaptation by Precise Detection of Metamodel Changes. In *In Proc. of ECMDA 2009*, pages 34–49, Enschede,, Pays-Bas, 2009. Springer.

[21] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Managing model adaptation by precise detection of metamodel changes. In *ECMDA-FA*, pages 34–49, 2009.

[22] H. Giese and R. Wagner. Incremental model synchronization with triple graph grammars. In *Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genova, Italy*, 2006.

[23] ATLAS Group. Atlas Transformation Language Project Website. `http://www.eclipse.org/m2m/atl/`, 2007.

[24] Boris Gruschko, Dimitrios S. Kolovos, and Richard F. Paige. Towards synchronizing models with evolving metamodels. In *Proc. Workshop on Model-Driven Software Evolution (MODSE), 11th European Conference on Software Maintenance and Reengineering*, 2007.

[25] Markus Herrmannsdoerfer. Cope - a workbench for the coupled evolution of metamodels and models. In *SLE*, pages 286–295, 2010.

[26] Kenn Hussey and Marcelo Paternostro. Advanced features of EMF. Tutorial at EclipseCon 2006, California, USA. Available at: `http://www.eclipse.org/modeling/emf/docs/presentations/EclipseCon/EclipseCon2006_EMF_Advanced.pdf`, 2006.

[27] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720. ACM, 2006.

[28] Stuart Kent. Model driven engineering. In *Integrated formal methods*, pages 286–298. Springer, 2002.

[29] Maximilian Koegel, Jonas Helming, and Stephan Seyboth. Operation-based conflict detection and resolution. In *CVSM '09: Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 43–48. IEEE Computer Society, 2009.

[30] Dimitrios S. Kolovos. Extensible Platform for Specification of Integrated Languages for mOdel maNagement Project Website. `http://www.eclipse.org/epsilon`, 2014.

[31] Dimitrios S. Kolovos, Nicholas Drivalos Matragkas, Horacio Hoyos Rodriguez, and Richard F. Paige. Programmatic muddle management. In *XM@MoDELS*, pages 2–10, 2013.

[32] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. Rigorous methods for software construction and analysis. chapter On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages, pages 204–218. Springer-Verlag, Berlin, Heidelberg, 2009.

[33] Christian Krause, Johannes Dyck, and Holger Giese. Metamodel-specific coupled evolution based on dynamically typed graph transformations. In *ICMT*, pages 76–91, 2013.

[34] Philip Langer, Manuel Wimmer, Petra Brosch, Markus Herrmannsdörfer, Martina Seidl, Konrad Wieland, and Gerti Kappel. A posteriori operation detection in evolving software models. *Journal of Systems and Software*, 86(2):551–566, 2013.

[35] mbeddr team. mbeddr language workbench. `http://mbeddr.com/`, 2014.

[36] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm (extended technical report). Technical Report 2001-25, Stanford InfoLab, June 2001.

[37] David Méndez, Anne Etien, Alexis Muller, and Rubby Casallas. Transformation migration after metamodel evolution. In *Proc. ME Workshop*, 2010.

[38] Anantha Narayanan, Tihamer Levendovszky, Daniel Balasubramanian, and Gabor Karsai. Automatic domain model migration to manage metamodel evolution. In *MoDELS*, pages 706–711, 2009.

[39] B. Nuseibeh, S. Easterbrook, and A. Russo. Making Inconsistency Respectable in Software Development. *Journal of Systems and Software*, 56(11), 2001.

[40] Hamilton L. R. Oliveira, Leonardo Gresta Paulino Murta, and Cludia Werner. Odyssey-vcs: a flexible version control system for uml model elements. In *SCM*, pages 1–16. ACM, 2005.

[41] openArchitectureWare. openArchitectureWare Project Website. `http://www.eclipse.org/gmt/oaw/`, 2008.

[42] Richard F. Paige, Dimitrios S. Kolovos, and Fiona A.C. Polack. A tutorial on metamodelling for grammar researchers. *Science of Computer Programming, to appear*, 2014.

[43] Richard F. Paige, Dimitrios S. Kolovos, Louis M. Rose, Nikolas Drivalos Matragkas, and James R. Williams. Model management in the wild. In *GTTSE*, pages 197–218, 2011.

[44] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[45] John F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7), 1995.

[46] Louis Rose. Structures and processes for managing model-metamodel coevolution. PhD thesis, University of York, 2011.

[47] Louis M. Rose, Esther Guerra, Juan de Lara, Anne Etien, Dimitris S. Kolovos, and Richard F. Paige. Genericity for model management operations. *Software and System Modeling*, 12(1):201–219, 2013.

[48] Louis M. Rose, Markus Herrmannsdoerfer, James R. Williams, Dimitrios S. Kolovos, Kelly Garcés, Richard F. Paige, and Fiona A. C. Polack. A comparison of model migration tools. In *MoDELS (1)*, pages 61–75, 2010.

[49] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. Model migration with epsilon flock. In *Proceedings of the Third International Conference on Theory and Practice of Model Transformations*, ICMT'10, pages 184–198, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, Fiona A.C. Polack, and Simon Poulding. Epsilon Flock: a model migration language. *Software and Systems Modeling*, 13(2):735–755, 2014.

[51] Stephan Roser and Bernhard Bauer. Automatic generation and evolution of model transformations using ontology engineering space. *Journal on Data Semantics XI*, 11:321764, 2008.

[52] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[53] Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Model transformations. In *Formal Methods for Model-Driven Engineering - 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*, pages 91–136, 2012.

[54] Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. A methodological approach for the coupled evolution of metamodels and atl transformations. In *ICMT*, pages 60–75, 2013.

[55] Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. Automated coevolution of gmf editor models. In *SLE*, pages 143–162, 2010.

[56] Douglas C. Schmidt. Guest editor's introduction: Model-driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[57] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[58] Perdita Stevens, Jon Whittle, and Grady Booch, editors. *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings*, volume 2863 of *Lecture Notes in Computer Science*. Springer, 2003.

[59] Gabriele Taentzer, Florian Mantz, Thorsten Arendt, and Yngve Lamo. Customizable model migration schemes for meta-model evolutions with multiplicity changes. In *MoDELS*, pages 254–270, 2013.

[60] The Eclipse Foundation. MDT/OCL Project. http://projects.eclipse.org/projects/modeling.mdt.ocl. [Online; accessed 04-July-2014].

[61] James R. Williams, Richard F. Paige, and Fiona A. C. Polack. Searching for model migration strategies. In *Proceedings of the 6th International Workshop on Models and Evolution*, ME '12, pages 39–44, New York, NY, USA, 2012. ACM.