Toward an Automated Co-Evolution for Meta Attack Languages and Models

Andreas Siljefors, Simon Hacks

Stockholm University, Stockholm, Sweden

Abstract

The rapid evolution of software systems often necessitates corresponding changes in their underlying models. However, ensuring these models remain consistent with their evolving meta-models is a complex challenge within Model-Driven Engineering (MDE). This work addresses this challenge by adapting and implementing a model co-evolution approach within the Meta Attack Language (MAL), a domain-specific language for modeling cyber threats.

The primary goal of this research is to develop a prototype system that can automatically generate migration strategies to restore or improve the conformance between models and their meta-models following meta-model changes. It utilizes a multi-objective evolutionary algorithm (NSGA-II) to explore the search space for possible edit operations. The system was designed to prioritize correctness and reduce manual effort while accommodating requirements drawn from a design process backed by a conceptual framework developed from subsequent literature.

A series of controlled experiments evaluated the prototype system against several models derived from the MAL language coreLang. The results demonstrate that the system does provide some effort in reestablishing model conformance, although with some performance variability.

Our work contributes to the field of MDE by providing insights into the practical application of model co-evolution techniques in a domain-specific context. The developed prototype is a foundation for future research to refine these techniques and expand their applicability to a broader range of modeling scenarios.

Keywords

Model-Driven Engineering, Model Co-evolution, Domain Specific Languages, Meta Attack Language

1. Introduction

Cyber security is a vital part of today's society, and the need for protection against different threats and attacks is critical. Consequently, to ensure that systems and infrastructure fulfill this requirement, assessment is needed [1]. A common approach to this task is to simulate attacks in so-called attack simulations utilizing attack graphs. An attack graph depicts dependencies between different steps a potential attacker may take to compromise a system. As such, attack simulations can be seen as performing a set of staged penetration tests on a system to test its viability [2]. However, attack graphs for actual use cases often explode in complexity and size [1]. Thus, separating generic features from more domain-specific features may accelerate the creation of attack graph models and reduce the repetitiveness of workloads.

MAL (Meta Attack Language) is a framework that was developed with this concept in mind. As a metalanguage, MAL specifies domain-agnostic rules and logic for developing DSLs (Domain Specific Languages). MAL also automatically generates an attack graph based on the underlying design [2]. Hence, common rules and generic principles can be reused across various domains and use cases.

Since the suggestion of MAL [1], new DSLs have been developed upon the MAL framework, and an ecosystem of languages has started to form. However, as the number of DSLs increases, new additions and updates may result in unanticipated consequences for existing DSLs. Similar to structural changes in conventional software, changes to one feature may impact others. Therefore, DSLs must be adapted and updated when additions, deletions, or changes occur [3]. Meta-models are inherently prone to

Companion Proceedings of the 17th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling Forum, M4S, FACETE, AEM, Tools and Demos co-located with PoEM 2024, Stockholm, Sweden, December 3-5, 2024

andreas.siljefors@gmail.com (A. Siljefors); simon.hacks@dsv.su.se (S. Hacks)

0000-0003-0478-9347 (S. Hacks)

change, which in turn causes issues as existing models lose conformance to the new meta-model version, an issue referred to as the *domain evolution problem* [4]. The issue is addressed by migrating existing models to have them conform with the underlying meta-model [5]. The migration process is possible by manually identifying and collecting each change and then updating the model. However, as every change of the meta model has to be addressed manually, it is a process that would require a great deal of effort from the modeler [6, 7]. Evolution processes comprise many general changes; for instance, between UML (Unified Modeling Language) version 1.5 and 2.0, 238 elements were either modified, added, or deleted [8]. Thus, it is easy to see that there is a need for support from model co-evolution.

Moreover, reestablishing model and meta-model coherence is a well-known topic in research. It is called model co-evolution, and extensive research has been conducted on the subject [6, 9, 7]. Thus, a broad range of different approaches for meta-model evolution and model co-evolution exists. Approaches that, to varying degrees, automate the process since the manual reestablishment of coherence or conformance often is complex and prone to errors [10]. A more nuanced understanding of the various techniques is essential to choose an approach to apply and adapt to a specific case from this collection of approaches. This understanding can be obtained in the systemic literature review of model evolution and model co-evolution found in recent research conducted by Hebig et al. [7]. In their research, they describe it as a step-wise procedure, which starts by collecting changes, followed by identifying the changes captured, which, when identified, are used to support resolving models. To this procedure, Hebig et al. also describe the available approaches suggested for the different steps of the procedure and arguments for when they are appropriate.

Recent research [7] highlights 31 different approaches developed to simplify the process of model co-evolution in one way or another. The concept of meta-model evolution and model co-evolution is discussed in more detail below, as these concepts form the conceptual framework for this research.

As a meta language, this issue also relates to the MAL language. However, this problem has not yet been transferred to the context of the MAL domain. Moreover, in their article about the development and maintenance of languages in the MAL domain, Hacks and Katsikeas [11] discuss the need for additional tools that may support and reduce the effort for developers, such as automatic code completion and refactoring capabilities. Thus, there is a need for additional research on how model co-evolution techniques could be applied to MAL models and meta-models—a problem that this work intends to address.

In our work, we rely on the foundations of existing research. However, existing approaches differ in how they identify meta-model changes and how the instantiated model is migrated [7]. Also, there is a trade-off between automation and correctness. As automation helps with scaling workloads and reduces tedious manual work, approaches that rely more on automation also tend to produce models that, to a lesser extent, conform with new meta-model versions [7]. Thus, there are aspects to consider before deciding on an approach.

In this research, we address the challenges concerning the domain evolution problem related to MAL models by adapting and applying an approach for model co-evolution for this context. Following the approach by Kessentini et al. [3], we view model co-evolution as a multi-objective optimization problem. Moreover, we develop a prototype system, artifact, for migrating non-conforming MAL models implementing the approach suggested by Kessentini et al. [3]. Finally, to evaluate our artifact, we establish two main objectives:

- O1 Adherence to Baseline Sequences: Edit operations of candidate solutions are to adhere to the edit operations of the corresponding baseline sequences derived from the manually performed migrations with at least 60% correctness.
- O2 Correctness Ratio of Edit Operations: The ratio of correct edit operations of the candidate solutions is to be at least 60% to the total number of edit operations in the candidate solutions.

A 60% adherence threshold to baseline sequences was chosen to balance automation efficiency and correctness, allowing flexibility in the algorithm's alternative solutions while maintaining alignment with manual migrations. Similarly, the threshold for edit operation correctness was set to allow meaningful results during early prototype stages, recognizing the difficulty of achieving full accuracy.

The paper is structured as follows: We introduce the theoretical background, providing an overview of the Meta Attack Language and relevant literature concerning model co-evolution. Next, we describe the research methodology applied in this study. Following this, we present the developed prototype, focusing on the approach and algorithms implemented. We then discuss the initial evaluation of the prototype, and finally, we conclude with key findings and insights for future work.

2. Background

2.1. Meta Attack Language

The Meta Attack Language (MAL) is a metalanguage to create DSLs for probabilistic threat modeling, used to evaluate the cyber security settings of systems [1]. MAL can be understood as a language that merges the features of UML class and object diagrams with probabilistic attack-defense graphs [12].

DSLs based on MAL share common terminology and notations for modeling different entities. Components identified within a domain are called assets [1], much like how classes are depicted in UML. Assets may comprise attack steps, which are the potential threats or attacks to which the assets are exposed. These attack steps can be linked with additional attack steps shaping a path. Attack steps are labeled either **OR**, meaning that only the compromising of a single parent attack step is needed for reaching this step, or **AND**, meaning that all parent attack steps need to have been compromised for the attack step to be reached. **OR** attack steps are denoted |, while **AND** attack steps are denoted **&**. The consequence or risk connected to an attack step are specified in brackets following the attack step definition; the risks relate to the CIA (confidentiality, integrity, and availability) triad and are denoted with any combination of the letters in the acronym CIA.

Moreover, MAL also comprises defenses that may exist within systems. Defenses are of boolean type; hence, they are denoted as either **TRUE** or **FALSE**, and thus, if denoted as **TRUE**, the defense can prevent attack steps from occurring. Attack steps may also be assigned with probability distributions, indicating how costly attack steps are to complete [1]. Listing 1 depicts a short code example visualizing the syntax of the MAL language.

Listing 1: Exemplary MAL Code

```
category System {
  asset Network {
    | access
      -> hosts.connect
  }

  asset Host {
    | connect
      -> access
    | authenticate
      -> access
    | guessPassword
      -> guessedPassword
    | guessedPassword [
       Exponential(0.02)]
      -> authenticate
    & access
  }

  asset User {
    | attemptPhishing
      -> phish
    | phish [Exponential
       (0.1)]
      -> passwords.obtain
  }

  asset Password {
    | obtain
      -> host.authenticate
  }
}

associations {
  Network [networks] * <--
     NetworkAccess --> * [
     hosts] Host
  Host [host] 1 <--
     Credentials --> * [
     passwords] Password
  User [user] 1 <--
```

```
        Credentials --> * [
        passwords] Password
36  }
37
38  associations {
39    Network [networks] *
40    <-- NetworkAccess -->
41      * [hosts] Host
42    Host [host] 1
43      <-- Credentials -->
44      * [passwords] Password
45    User [user] 1
46      <-- Credentials -->
47      * [passwords] Password
48  }
```

2.2. Meta Model (Co-)Evolution

In Model-Driven Engineering (MDE), meta-models define the abstract aspects of modeling languages, specifying how models are created. Meta-models are typically represented using UML class diagrams, and the Meta-Object Facility (MOF) is a common standard, organizing meta-models into a 4-layer hierarchy. The need for model co-evolution arises when models lose conformance and must adapt to changes in their underlying meta-models. Various approaches exist to handle co-evolution, from manual transformation strategies to automated learning and constraint-based searches. In recent literature, extensive classification of existing approaches has distinguished 31 different techniques [7]:

Resolution Strategy Languages. These approaches comprise transformation languages to specify resolution strategies manually.

Resolution Strategy Generation. This category refers to approaches that generate partial strategies for meta-model changes.

Predefined Resolution Strategies. Includes approaches that apply predefined resolution strategies to achieve automation.

Resolution Strategy Learning. Approaches that aim at learning resolution strategies created by users, applying and accumulating strategies, thus reducing the workload of creating strategies over time.

Constrained Model Search. This includes approaches that utilize a constraint-based search to search for a space of valid model alternatives based on the initial model and new meta-model version. Hence, they are not dependent on specific meta-model changes, and they also support partial automation of model resolution per model and not per change.

Identify Complex Changes. Comprise approaches that focus on change identification, more specifically, complex changes.

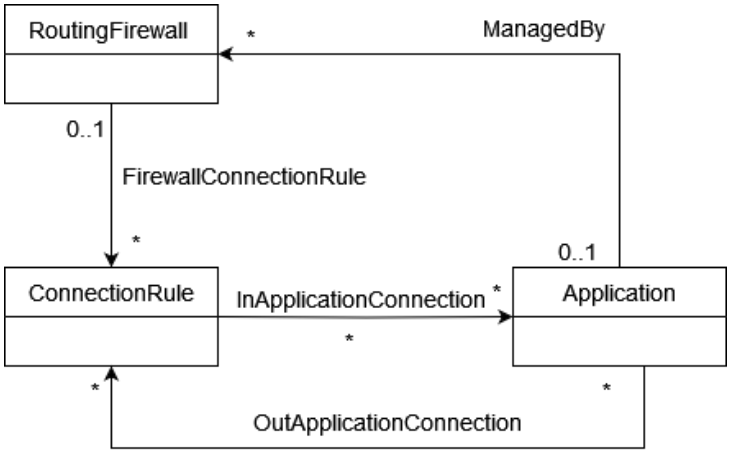

Figure 1: Example of a meta-model

Figure 1 shows an example of a meta-model which is instantiated as the model shown in Figure 2. Besides instantiating the *RoutingFireWall*, and the *ConnectionRule* classes the model has three

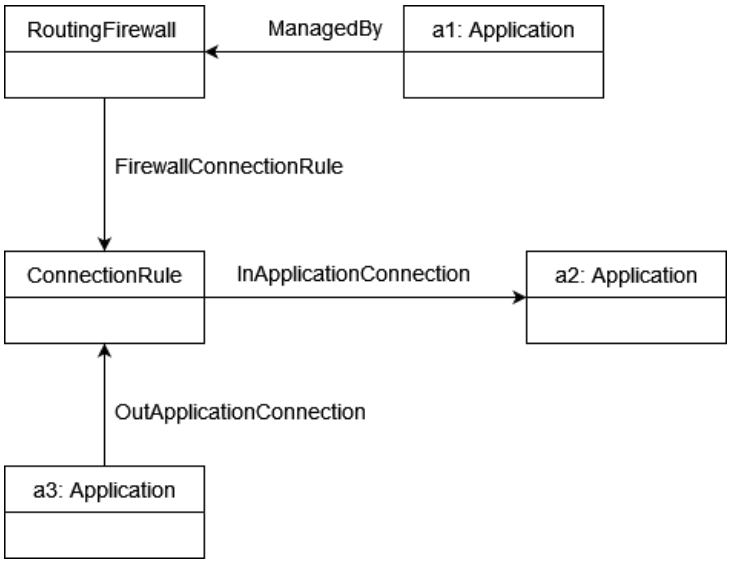

Figure 2: Example of a model instantiation

instantiations of the *Application* class. Meta-model evolution refers to changes applied to a meta-model, including adding new elements, modifying existing ones, or simply deleting elements. These changes are either atomic or complex. Atomic changes only comprise one element, such as deleting or renaming an element.

In contrast to atomic changes, complex changes, such as moving one property of an element to another, remove the property from one element and add it to another. Hence, a complex change comprises a set of atomic changes [8, 7]. As a set of actions, understanding the editing action to be performed on one element, one also knows the editing action for the other element [8]. Hebig et al. [7] illustrate this relationship in the following way: a move property action includes the atomic changes delete property and add property. By moving the value from the delete to the add property, the change is performed accordingly. However, the added property change could not be instantiated correctly without awareness of the complex change.

Based upon an extensive catalog of operators [13], recent literature has identified the potential operations that can be performed on model elements to modify them [3]. These operations are presented in Table 1.

Table 1
Change operators [3]

Change Operator	Description
Create/Delete	Add or remove an element in the initial model.
Retype	Replace an element by another equivalent element having a different type.
Merge	Combine several model elements of the same type into a single element.
Split	Divide a model element into several elements of the same type.
Move	Transfer an attribute from one model element to another.

Figure 3 depicts a changed version of the meta-model in Figure 1. The change comprises the steps: extract sub-classes (*Managing application, Server side application, and Client side application*), then the *Application* class is made abstract, and lastly, the references *InApplicationConnection and OutApplicationConnection* are edited.

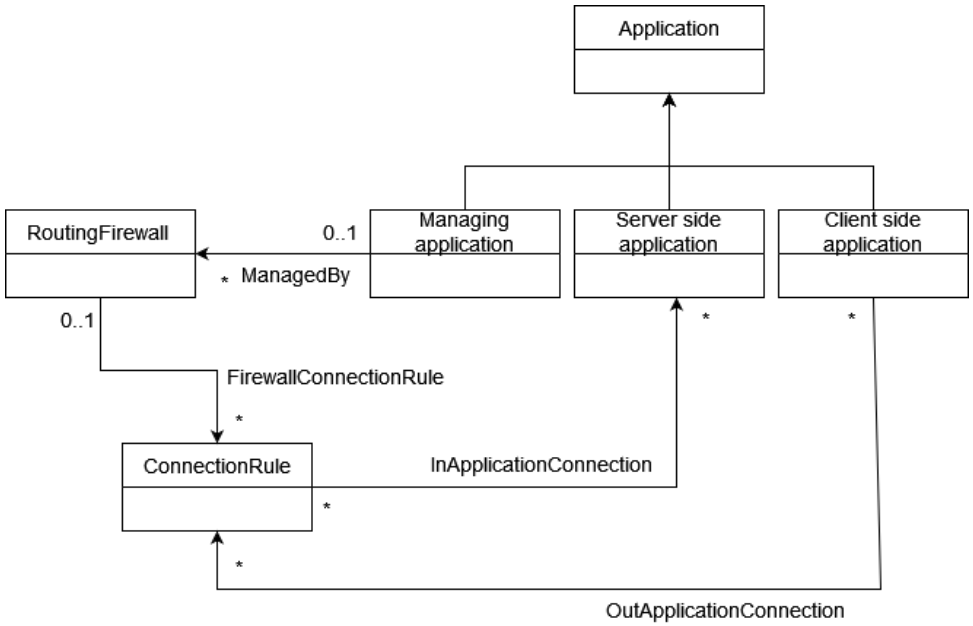

Figure 3: Example of an evolved meta-model

As the meta-model now has three different sub-classes of application the model instantiantion need to be updated to reflect these changes. Figure 4 shows a visual example of a resolved model, where the model has the correct *Application* class instances.

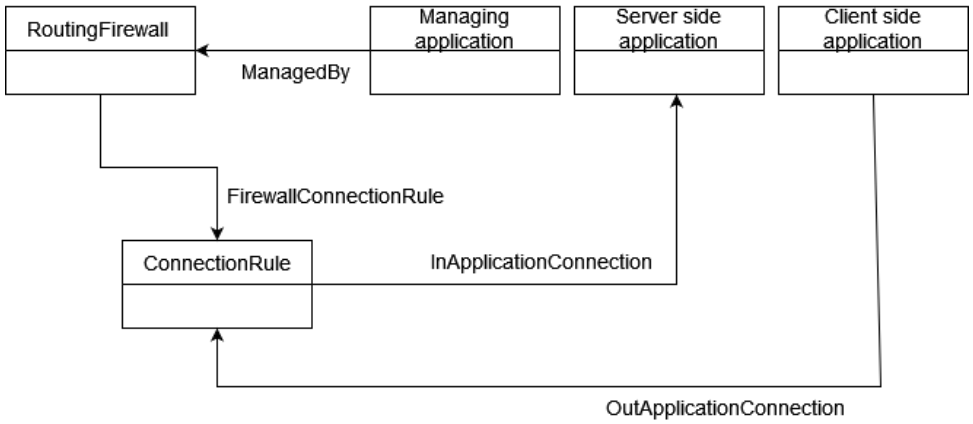

Figure 4: Example of a co-evolved model

For model migration processes, changes in meta-models need to be collected. This can be done either offline or online. Offline change collection involves comparing two versions of the meta-model, the original and the changed, after modifications have occurred, independent of the tools used to make these changes. In contrast, the online change collection tracks changes as they happen and presents them in chronological order [7].

When changes have been collected, the migration process may continue with identifying the changes. This mainly refers to outlying all the different atomic and complex changes with their corresponding types applied to meta-models. Information on atomic changes is provided as a pair of meta-model versions; more specifically, it is the difference between two meta-model versions, and as mentioned, it is needed for offline change collection.

The process culminates in the migration, using the collected and identified changes. This step comprises two main concepts, *resolution strategy*, which indicates how, for a specific change, models are to be changed. The other concept is *resolution technique*, which refers to how resolution strategies

are created and applied.

Changes in meta-models may occur more or less independently. Hence, changes will occur in a certain chronological order. To resolve changes, one way is to proceed in the same order as the changes occurred. However, the chronological order information is unavailable when changes are collected through *offline change collection*. Thus, resolving changes collected by this strategy is done randomly or by establishing an optimal resolution order to minimize the manual effort.

Concerning manual effort, specifying a strategy is a significant aspect to consider. A classification system based on a ratio between a user's invested effort in specifying a resolution strategy on the one side and the range of models that can be resolved with the chosen strategy on the other has been promoted in recent research [7]. The ratio classifications described are 1) 1:1, which concerns situations where every model requires manual intervention such as selection or configuration of strategy; 2) 1:n, concerning cases for which one initial decision, e.i. strategy selection is needed and then applied to a range of models; 3) 0:n, which relates to cases where reuse of existing strategies, uncoupled to specific meta-models, are applied, meaning no manual intervention is required. These three classifications have been labeled *benefit classes* [7].

The benefit classes discussed above are used to understand the automation capabilities of approaches. Approaches that belong to benefit class 0:n require no manual involvement of model resolution. A few approaches from the category *Resolution Strategy Generation* are found in this benefit class [7], the approach suggested by de Geest et al. [14] to mention one. Moreover, the approaches of *predefined resolution strategies* are all considered as benefit class 0:n. They may support automation by multiple solutions, operator-based approaches, or by a single solution strategy, non-operator-based approaches [7]. These approaches include COPE suggested by Herrmannsdoerfer et al. [15], ASIMOV by Florez et al. [16], and approaches by Cicchetti et al. [5], and Gruschko et al. [17].

Benefit class 1:n includes approaches that require a human-made decision for a meta-model change to select a strategy to co-evolve many models [7]. Three main strategies have been identified to do this [7]. The first is *strategy specification*, which refers to modelers freely specifying a strategy. This strategy complements all *resolution strategy languages* classification approaches. Which includes approaches such as the visual language suggested by Sprinkle et al. [4] and the approach of using transformation languages suggested by Wimmer et al. [18]. Additionally, it is also supported by the *resolution strategy generation* approach by Meyers et al. [19], and some *predefined resolution strategies*, including Gruschko et al. [17], and COPE [20].

The second strategy mentioned, is referred to as *Free Strategy Manipulation*, it allows for free manipulation of default strategies [7]. Four approaches of *resolution strategy generation* are suggested to support this strategy, including the approaches of de Geest et al. [14] and Meyers et al. [19]. Lastly, *Strategy Configuration*, allow for some restricted configuration based upon a set of fixed parameters [7]. This strategy is supported by some of the *predefined resolution strategies* approaches, such as the ones suggested by Cicchetti et al. [5] and Gruschko et al. [17].

As discussed above, benefit class 1:1 requires human intervention for every model to be resolved. Four approaches that support *strategy configuration* for this benefit class have been found [7], including the approaches by Meyers et al. [19], and Cicchetti et al. [5]. Moreover, the approaches of the classification *constrained model search* specialize in assisting manual migration by generating alternative model versions based on model constraints. Thus, they are also found in this benefit class [7].

3. Research Method

This research employed a design science methodology, specifically adopting the Systems Development Research Methodology (SDRM) proposed by Nunamaker et al. [21]. SDRM follows an iterative process, including the steps of (1) Constructing a Conceptual Framework, (2) Developing a System Architecture, (3) Analyzing and Designing the System, (4) Building the Prototype System, and (5) Observing and Evaluating the System.

The research developed a conceptual framework grounded in the principal problem of the research

and the research goal which were contextualised by a comprehensive literature review and document study, and the desired functionality. A visual representation of this framework is shown in Figure 5.

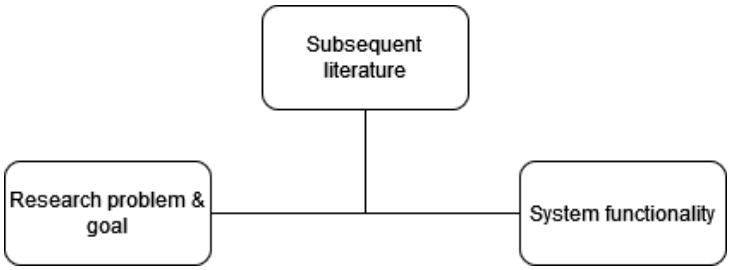

Figure 5: Conceptual framework

The identified approaches were analyzed and contextualised with used for the developed architecture to elicit system requirements and objectives. The requirements for the prototype system were established to guide the design and implementation phases:

- R1 The prototype system should automate the process of producing a migration strategy that, when applied, improves the conformance of the input model w.r.t the meta-model. A prototype system shall help reduce the manual effort.
- R2 The co-evolution approach of the prototype system should support benefit class 1:1. As the principal emphasis is finding an adequate migration strategy, the co-evolution approach does not need to resolve the model by itself automatically.
- R3 The prototype approach should focus on analyzing static changes. Collecting change history is not a priority in this research.
- R4 The prototype system's model co-evolution approach should be known to have been implemented. By using a tested approach, technological risks are reduced.

The third phase (Analyzing and Designing the System) involved designing the artifact. It primarily relied on brainstorming and convergent thinking to select a final design based on feasibility and time constraints. The system architecture evolved iteratively as new insights were gained, resulting in adjustments to the solution.

During the implementation phase, the aim was to produce prescriptive knowledge. The model representation included algorithms, system functionality, and system operation design principles. The artifact was then implemented according to the system architecture.

The evaluation was based on the objectives and focused on the algorithms' effectiveness in restoring conformance between a MAL metamodel and model instantiations. Due to the use of randomized algorithms, the evaluation complied with the guidelines of Arcuri and Briand [22], which recommend assessing randomized algorithms through many iterations. The evaluation used a legacy version of MAL DSL coreLang [23] (version 0.5.0) and its current version (version 1.0), along with 10 manually migrated models, following the approach of Kessentini et al. [3]. At least 30 iterations of computer-based simulations were conducted, and data was collected from computer logs. Precision and recall measures were used to benchmark the results (cf. Section 5.

4. A Prototype for Co-Evolution in MAL

The model co-evolution strategy proposed by Kessentini et al. [3] was selected as the most suitable approach for the problem. Their method treats co-evolution as a multi-objective search problem to identify the optimal sequence of edit operations. This sequence is used to revise an initial model to conform to an updated version of the corresponding meta-model.

The approach is framed as exploring a search space composed of all possible sequences of edit operations applied to the initial model. The search aims to find the best sequence by evaluating key objectives.

The potential number of solutions in a search space is often large, as a candidate solution can involve any combination of available operations. To handle this complexity, the NSGA-II algorithm is employed [24]. NSGA-II is an elitist multi-objective evolutionary algorithm designed to sort and prioritize non-dominant solutions, favoring candidates that perform better based on predefined objectives. The pseudo-code for NSGA-II is shown in Algorithm 1.

The algorithm begins by generating an initial population of solutions called the parent population. It then applies variation operators such as binary tournament selection, recombination, and mutation to generate a child population from the parents. The operators introduce randomness and diversity into the selection process.

The next step involves combining the parent and child populations to form a larger group of solutions, which is then evaluated. NSGA-II uses a crossover operator to split parent solutions at a random point or points and swap their parts to create new combinations. Mutation operators randomly modify one or two operations within a sequence, either by swapping operations or replacing one with another. Any redundant operations framed that may appear in a sequence are to be removed.

The combined population is sorted according to non-dominance, with the algorithm identifying and organizing non-dominant solutions into layers, starting from the first so-called Pareto front. This process continues until all solutions have been sorted. Each candidate solution is evaluated based on its performance and objectives.

To provide a practical example, we consider a JSON-based model, which we call *App model*, undergoing co-evolution. The initial *App model* contains four non-conforming elements that must be corrected to restore conformance to the meta-model[1]. However, these elements are first preprocessed to generate valid edit operations, defining the NSGA-II algorithm's search space.

With the search space established, NSGA-II generates an initial parent population P_0. The algorithm applies a one-point crossover and a mutation operator to produce an offspring population Q_0. The crossover operator splits the solutions of the parent population at a random position and swaps the sub-solutions to create new solutions. The mutation operator adds further randomness in the sequences by swapping operations or replacing one with a random alternative. The decision to apply one or both mutation techniques is random, and if a redundant operation appears, a repair operator removes it. Both populations produced are the same size and combined to form a larger population $R_t = P_t \cup Q_t$.

Once the combined population is formed, the solutions created from the initial *App model* elements are sorted based on non-dominance. This process compares each solution to determine if it is non-dominant, that is a solution which out-performs all other solutions. Non-dominant solutions are assigned to layers starting from the first Pareto front, and the sorting continues until all solutions have been categorized. The comparison is based upon three objectives: the number of non-conforming elements in the revised model, the number of operations in the sequence, and the coherence between the initial and revised models concerning the overall fitness regarding the *App model*.

The first objective is to minimize non-conformity, measured by the number of broken constraints in the revised *App model*. For a given solution sequence *s*, the function $f1(s) = nvc(s)$ calculates the number of violations. The second objective is to minimize the number of edit operations applied to the model, represented by the function $f2(s) = nbOp$, which calculates the sequence length. The third objective assesses the dissimilarity between the initial and revised models, calculated by the function $f3(s) = dis(s)$. This function measures the differences between the two models after the sequence of edit operations is applied. Relying on model element types may be unreliable due to changes in the meta-model. An inconsistency between the models is calculated based on whether elements have been added or deleted, which evaluates the proportion of common elements between the initial and revised models.

In the described procedure, the co-evolution process using NSGA-II finds an optimal solution for *App model* that balances the competing objectives of minimizing non-conformity, edit sequence length, and dissimilarity between the initial and revised model versions. The algorithm iteratively refines candidate solutions, navigating the complexity of the search space to produce an updated *App model* that aligned

[1]https://drive.google.com/drive/folders/13ORCJcgyhutWO8yVEGiCED1gB9H$_W h7z?usp = sharing$

with the updated meta-model.

Algorithm 1 NSGA-II

procedure NSGA-II
  Create an initial population P_0
  Create an offspring population Q_0
  $t = 0$
  while stopping criteria not reached **do**
    $R_t = P_t \cup Q_t$
    $F =$ fast non-dominated-sort(R_t)
    $P_{t+1} = \emptyset$ and $i = 1$
    while $|P_{t+1}| + |F_i| \leq N$ **do**
      Apply crowding-distance-assignment(F_i)
      $P_{t+1} = P_{t+1} \cup F_i$
      $i = i + 1$
    end while
    Sort($F_i, \prec_n$)
    $P_{t+1} = P_{t+1} \cup F_i[N - |P_{t+1}|]$
    $Q_{t+1} =$ create-new-pop(P_{t+1})
    $t = t + 1$
  end while
end procedure

We produced an architecture for our artifact using NSGA-II as the main approach for conducting model co-evaluations, which is presented in Figure 6

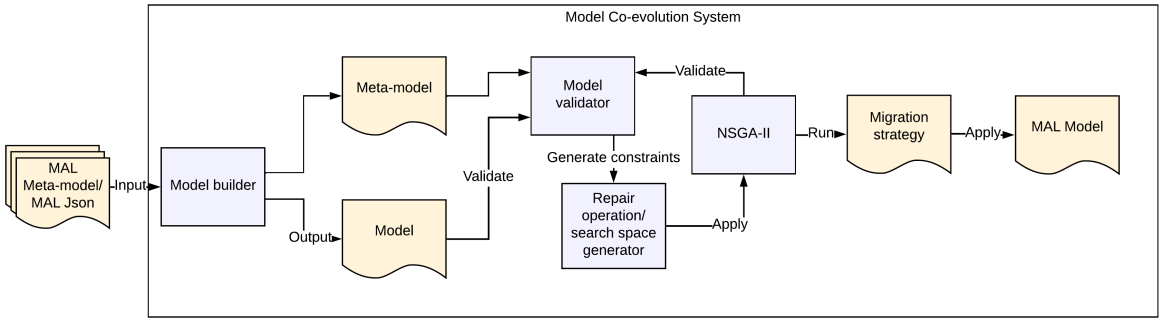

Figure 6: System architecture

5. A First Evaluation

The experimental evaluation was conducted to test the adapted approach under controlled conditions. The experiments involved iterative adjustment and assessment of the performance against predefined metrics. The results indicated some challenges and disparities.

A rough specification of the models applied in the experiment is presented in Table 2. The first row refers to the number of elements not present in the evolved meta-models, thus constituting the elements that need to be removed from the model to restore conformance. The baseline sequences represent the number of expected operations applied to the models to eliminate non-conformities and uphold model structure regarding the new meta-model version. These sequences were derived from

a manual migration process conducted for each model. The sequence only concerned creating and deleting operations, as only those two operation types were considered relevant for the models used for the experiment. Additionally, the modeling elements are concerned with the number of elements found in each model. That is the total number of objects, attributes, and references.

Table 2
Model specification

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
No. non-conforming elements	4	2	3	3	2	4	3	4	2	4
Baseline sequence	7	4	6	6	4	8	6	8	4	9
Modeling elements	33	11	11	18	10	24	10	17	11	16

For the evaluation of the artifact, every model was tested 30 times, and for each iteration, the result of every metric was reported. The population size used for the experiment was set to 250, and the number of runs for NSGA-II to 150. Moreover, the crossover probability was set to 0.7 and the mutation probability to 0.3.

Table 3 and 4 outline the main results from the experiments, that is, the average performance for each model across all iterations. Each value in the table represents the average result of the specific metric, summarizing the system's ability to generate accurate and effective migration strategies.

Table 3
The average precision & recall results for each model

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Std Dev
Precision	0.69	0.73	0.77	0.60	0.87	0.68	0.80	0.63	0.76	0.67	0.078
Recall	0.70	0.64	0.72	0.62	0.60	0.61	0.67	0.61	0.75	0.75	0.056

Table 4
The average Nvc & NoOp results for each model

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
No. violated constraints (Nvc)	0.38	0.45	0.20	0.33	0.10	0.23	0.0	0.28	0.05	0.20
No. total operations (NoOp)	7.0	3.4	4.8	6.3	3.0	7.2	5.1	7.8	4.0	10.0

The precision metric measures the proportion of correct operations within the candidate solutions generated by the system. Precision values across the models ranged from 60% to 87%, with some models (e.g., M5) exhibiting high precision, indicating that the system accurately selected relevant operations. However, lower precision in models such as M4 and M8 suggests that the system occasionally introduced unnecessary or incorrect operations, which did not align with the expected baseline sequences.

Recall assesses the system's ability to identify all necessary operations for model migration. The recall values calculated from the iterations vary from 60% to 79%. This demonstrates that the system, on a general level, performed well in finding essential operations. However, some models had lower recall scores, indicating that the system sometimes failed to recognize operations of the baseline sequences.

The Nvc tracks the number of constraints violated in the final model after applying the migration strategy. In Table 4, Nvc shows the ratio of violated constraints to the initial number of nonconforming elements. That is, it shows relative improvement in the number of violated constraints after applying the migration strategy. Lower Nvc values indicate a higher level of conformance, i.e., fewer violations. The

general performance did not reach initial expectations. The system accurately found non-conforming elements, with some models (e.g., M7 and M9) showing very few or no violations. However, the M1, M2, M4, M6, and M8 models had higher Nvc values, suggesting that the generated migration strategies were less effective in these cases. Since these were larger models, it may indicate that model size harmed performance. In general, although the system did partly restore the conformance of models, it did not achieve a solid performance in this regard.

The NoOp metric represents the total number of operations a candidate solution applies. The variation in NoOp values across different models reflects each model's complexity, the baseline sequences' size, and the operations required to achieve conformance. The results vary slightly but generally reflect the number of elements in the baseline sequences. However, the result has outliers (e.g., M3) for which the candidate solution was less than one element on average.

To put the results of the experiments conducted into context, Figure 7 provides a bar chart of the average precision and recall for all models evaluated in this study, and the average result reported by Kessentini et al. [3]. The experiments carried out for this research did not achieve results with the same magnitude as Kessentini et al. [3]. They report an average precision of 87%, and an average recall of 89%. The precision average of this study reached 72%, and the recall average 67%.

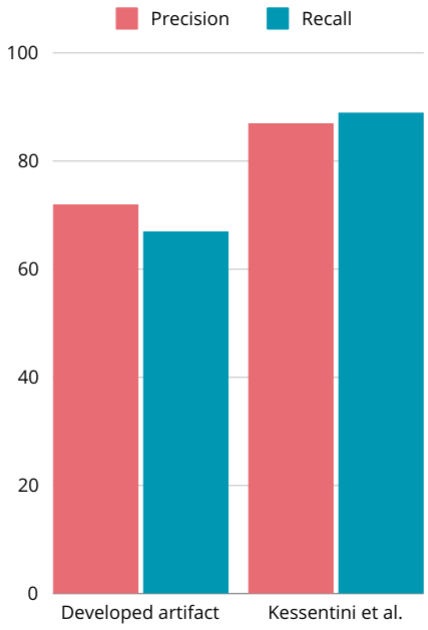

Figure 7: Average results by implementation

The experimental evaluation results provide insights into the performance of the prototype system. Overall, the system demonstrated the capability to reestablish conformance between models and meta-models, particularly concerning recall, as it reached the system objectives' thresholds. However, it showed notable challenges in achieving higher precision and reducing the number of violated constraints. Also, compared to the results reported by Kessentini et al. [3], the results of this study fall short. These findings suggest that while the prototype is a promising step towards automating model co-evolution, further refinement is needed to improve accuracy and efficiency, particularly in handling complex models.

Regarding threats to validity, algorithmic parameters—such as mutation and crossover rates and the number of runs—may have impacted the results, along with potential human error in evaluating baseline sequences. These factors were monitored, with parameter values transparently reported to

facilitate future comparisons.

6. Conclusions

In this paper, we address the challenges of model co-evolution in the context of the MAL, focusing on maintaining conformance between models and evolving meta-models. Following a design science research approach, a prototype system was developed to generate migration strategies using the NSGA-II algorithm automatically. The system successfully restored model conformance, though performance varied across different models, with larger models posing a greater challenge. Concerning the first objective outlined, the results show varying levels of adherence to the baseline sequences across different models. Precision values, which measure the proportion of correct operations within the candidate solutions, ranged from 60% to 87%. This indicates that the candidate solutions generated by the system generally aligned well with the baseline sequences. Overall, the system demonstrated an ability to produce candidate solutions that adhered to the baseline sequences at the threshold of 60%. The prototype system also successfully met the second system objective regarding the correctness ratio. As this ratio for the produced candidate solutions by the prototype system ranged from 60% to 87% across the models, the 60% threshold was exceeded by all models.

Despite meeting the system objectives, limitations in handling larger models and deviations from the original NSGA-II implementation suggest further refinement. Moreover, algorithmic parameter settings may have influenced the outcomes. Additionally, human error or bias could potentially affect baseline sequences used for evaluating candidate solutions. While efforts were made to minimize these risks through careful documentation and consistent processes, some bias may remain.

Future research could refine the evaluation using a broader range of models or more accurate baseline sequences, possibly applying meta-model matching to extract precise differences across model versions. Attention could also be directed to improving evaluation metrics and enhancing the system's ability to manage complex changes.

References

[1] P. Johnson, R. Lagerström, M. Ekstedt, A meta language for threat modeling and attack simulations, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES '18, Association for Computing Machinery, New York, NY, USA, 2018. doi:10.1145/3230833.3232799.

[2] S. Katsikeas, S. Hacks, P. Johnson, M. Ekstedt, R. Lagerström, J. Jacobsson, M. Wällstedt, P. Eliasson, An attack simulation language for the it domain, in: H. Eades III, O. Gadyatskaya (Eds.), Graphical Models for Security, Springer International Publishing, Cham, 2020, pp. 67–86.

[3] W. Kessentini, H. Sahraoui, M. Wimmer, Automated metamodel/model co-evolution using a multi-objective optimization approach, in: Modelling Foundations and Applications, Springer International Publishing, Cham, 2016, pp. 138–155.

[4] J. Sprinkle, G. Karsai, A domain-specific visual language for domain model evolution, Journal of Visual Languages & Computing 15 (2004) 291–307. doi:10.1016/j.jvlc.2004.01.006.

[5] A. Cicchetti, D. D. Ruscio, R. Eramo, A. Pierantonio, Automating co-evolution in model-driven engineering, in: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference, IEEE Computer Society, USA, 2008, p. 222–231. doi:10.1109/EDOC.2008.44.

[6] L. Rose, R. Paige, D. Kolovos, F. Polack, An analysis of approaches to model migration (2009).

[7] R. Hebig, D. E. Khelladi, R. Bendraou, Approaches to co-evolution of metamodels and models: A survey, IEEE Trans. Softw. Eng. 43 (2017) 396–414. doi:10.1109/TSE.2016.2610424.

[8] D. E. Khelladi, R. Hebig, R. Bendraou, J. Robin, M.-P. Gervais, Detecting complex changes during metamodel evolution, in: Advanced Information Systems Engineering, Springer International Publishing, Cham, 2015, pp. 263–278.

[9] M. Herrmannsdoerfer, G. Wachsmuth, Coupled Evolution of Software Metamodels and Models, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 33–63. doi:10.1007/978-3-642-45398-4_2.
[10] J. Schönböck, A. Kusel, J. Etzlstorfer, E. Kapsammer, W. Schwinger, M. Wimmer, M. Wischenbart, Care - a constraint-based approach for re-establishing conformance-relationships, in: Asia-Pacific Conference on Conceptual Modelling, 2014.
[11] S. Hacks, S. Katsikeas, Towards an ecosystem of domain specific languages for threat modeling, in: Advanced Information Systems Engineering, Springer International Publishing, Cham, 2021, pp. 3–18.
[12] W. Wideł, S. Hacks, M. Ekstedt, P. Johnson, R. Lagerström, The meta attack language - a formal description, Comput. Secur. 130 (2023). doi:10.1016/j.cose.2023.103284.
[13] M. Herrmannsdoerfer, S. Benz, E. Jürgens, Automatability of coupled evolution of metamodels and models in practice, 2008, pp. 645–659. doi:10.1007/978-3-540-87875-9_45.
[14] G. de Geest, S. Vermolen, A. van Deursen, E. Visser, Generating version convertors for domain-specific languages, in: 2008 15th Working Conference on Reverse Engineering, 2008, pp. 197–201. doi:10.1109/WCRE.2008.50.
[15] M. Herrmannsdoerfer, S. Benz, E. Juergens, Cope - automating coupled evolution of metamodels and models, in: S. Drossopoulou (Ed.), ECOOP 2009 – Object-Oriented Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 52–76.
[16] H. Florez, M. Sánchez, J. Villalobos, G. Vega, Coevolution assistance for enterprise architecture models, in: Proceedings of the 6th International Workshop on Models and Evolution, Association for Computing Machinery, New York, NY, USA, 2012, p. 27–32. doi:10.1145/2523599.2523605.
[17] B. Gruschko, D. Kolovos, R. Paige, Towards synchronizing models with evolving metamodels (2007).
[18] M. Wimmer, A. Kusel, J. Schönböck, W. Retschitzegger, W. Schwinger, On using inplace transformations for model co-evolution 711 (2010) 65–78.
[19] B. Meyers, M. Wimmer, A. Cicchetti, J. Sprinkle, A generic in-place transformation-based approach to structured model co-evolution, ECEASST 42 (2011). doi:10.14279/tuj.eceasst.42.608.
[20] M. Herrmannsdoerfer, Cope – a workbench for the coupled evolution of metamodels and models, in: Software Language Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 286–295.
[21] J. F. Nunamaker, M. Chen, T. D. M. Purdin, Systems development in information systems research, Twenty-Third Annual Hawaii International Conference on System Sciences 3 (1990) 631–640 vol.3.
[22] A. Arcuri, L. Briand, A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 1–10. doi:10.1145/1985793.1985795.
[23] Corelang, https://github.com/mal-lang/coreLang, 2024. Accessed: 2024-02-26.
[24] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, in: Parallel Problem Solving from Nature PPSN VI, Springer Berlin Heidelberg, 2000, pp. 849–858.

A. Online Resources

The example model/ meta-model discussed can be found at https://drive.google.com/drive/folders/13ORCJcgyhutWO8yVEGiCED1gB9H_Wh7z?usp=sharing

The prototype and the used data for the evaluation can be found at https://github.com/4ndsil/ModelCoEvolution.