**CO-EVOC: An Enterprise Architecture Model Co-Evolution Operations Catalog**

*Completed Research*

**Nuno Silva**  
University of Lisbon  
nuno.miguel@tecnico.ulisboa.pt

**Pedro Sousa**  
University of Lisbon  
pedro.manuel.sousa@tecnico.ulisboa.pt

**Miguel Mira da Silva**  
University of Lisbon  
mms@tecnico.ulisboa.pt

**Abstract**

Models are a fundamental aspect of enterprise architecture, as they capture the concepts and relationships that describe the essentials of the different enterprise domains. A meta-model, in model-driven engineering parlance, defines the rules to create and update such models. Over time, to keep up with the need to capture a more complex reality in their enterprise architecture models, organizations need to enrich the meta-model and upgrade the existing models accordingly. In this article, we present a catalog of operations that enables the automation of co-evolution of enterprise architecture models, driven by a set of meta-model changes. The operations presented refer to the evolution of the ArchiMate meta-model since it is commonly used in the design of business architecture models.

**Keywords**

Enterprise architecture, meta-model, model, co-evolution.

**Introduction**

The goal of the research presented in this paper is to reduce the effort of remodeling of Enterprise Architecture (EA) models whenever the meta-model (the language used to express these models) evolves. EA models are a fundamental aspect of EA initiatives because they are used to design and disseminate the enterprise’s organizational structure, business processes, information systems, and IT infrastructure (Lankhorst et al. 2013). Selecting and managing the set of rules of a domain-specific modeling language used to express EA models, are two critical aspects in EA initiatives (Armour et al. 1999; Espinosa and Armour 2008; Kaisler et al. 2005; Land et al. 2009). Since domain-specific modeling languages are typically defined at the very first steps of EA initiatives, they are likely to evolve in subsequent stages of EA initiatives. There are several causes for domain-specific modeling languages to evolve. Internal causes, when the needs of expressiveness increase along with the evolution and scope of the EA initiatives, and external causes, when the standards or compliance rules change.

Take the example of the ArchiMate (The Open Group 2013) modeling language. This standard has seen significant changes since its conception in 2010 up to its current 3.0.1 version (The Open Group 2013). New domains, concepts, and relationships were added while others were updated or removed from the language. This holds true to other EA specific modeling languages, either proprietary or open. The real challenge of evolving domain-specific modeling languages is the co-evolution of the models that might no longer comply with the new version of the language. The need to adapt the EA model combined with its inherent complexity puts a strenuous effort in organizations that seek to evolve and maintain their existing EA, leading ultimately to an ineffective EA process within organizations.

*Twenty-fourth Americas Conference on Information Systems, New Orleans, 2018*
These domain-specific modeling languages are defined and formalized as meta-models by the model-driven engineering community (Antonio Cicchetti et al. 2008a; Herrmannsdoerfer et al. 2011; Wachsmuth 2007). Taking into account the numerous changes a meta-model can have, one comes to the conclusion that not all the meta-model changes are problematic. In fact, a significant part of the meta-model evolution is just the addition of new concepts, and therefore have no implications for the existing EA models. Another part, however, is the redefinition of concepts in the meta-model, which typically happens when organizations need to increase the modeling capacity and expressiveness of their EA models. In spite of being a limited part of meta-model changes, these last changes are usually responsible for most of the effort implied in the evolution of the meta-model, because they also force changes to the existing models to match the new meta-model.

Based on over a decade of EA projects in both public and private enterprises from practically all industry sectors, we observed that organizations see multiple advantages of starting their EA initiative with a simpler meta-model and later, as they gain more maturity in these initiatives, enrich it according to their needs. However, when faced with the enormous effort that needs to be made to update existing models when the meta-model evolves, they often abandon the idea of starting with a simple meta-model and choose to start with a complex, difficult-to-explain and understand meta-model. Such a choice often leads to the failure of EA initiatives due to difficulties of dissemination to the whole organization. So, this research is relevant because it allows alternative approaches to EA initiatives. But this research is even more pertinent because, in practice, standards themselves change faster than organizations are able to implement EA initiatives, forcing forced changes to initial models. So, organizations do not really have a choice but to be able to evolve their EA meta-model and update their models accordingly.

In this paper, we propose a catalog of fully automated EA model co-evolution operations with the goal of reducing the overall effort and error-proneness of transforming EA models according to the changes done in the EA meta-model.

The structure of the paper is as follows. The “Research Problem” section formalizes the problem of model co-evolution. Then, “Related Work” presents a set of proposed approaches for solving the co-evolution problem. “Research Proposal” describes our research proposal as well as the theory supporting our contribution. The “Demonstration” section presents a real-life scenario in which co-evolution operations from the proposed catalog were applied to solve the EA model co-evolution problem in a governmental organization. We conclude the paper and refer to future efforts in “Conclusion”.

**Research Problem**

In EA context, meta-models and models are also the targets of the evolutionary pressure with meta-model changes often occurring due to the iterative nature of the meta-model and model construction process (Florez et al. 2012). Changes to the meta-model have a high probability of impacting all models that conform to it. Depending on the organization’s EA maturity level, it can lead to a high manual remodeling effort which leads to inconsistencies between the new version of the meta-model and its respective models.

The co-evolution problem occurs when an element from the meta-model changes and the model no longer conforms to the new meta-model, as Figure 1 illustrates. After performing an evolution Δ of a meta-model MM into MM’, the goal is to co-evolve model m that conforms to MM, to m’ that conforms to MM’, by applying a set of model transformations T that are aligned with evolution Δ (Antonio Cicchetti et al. 2008b; Florez et al. 2012).

Co-evolution of models is strictly related to the notion of information preservation (Wachsmuth 2007) from which the possible meta-model changes can be distinguished into additive, subtractive, and refactoring. Therefore, changes that occur on a meta-model may have different effects on the related models. These changes are classified as follows (Antonio Cicchetti et al. 2008a, 2008b):

- **Non-breaking changes.** Model conformance to a meta-model is preserved when changes are made to the meta-model;
- **Breaking and resolvable changes.** Model conformance to a meta-model is broken when changes are made to the meta-model, however, they can be automatically resolved;
- **Breaking and unresolvable changes.** Model conformance to a meta-model is broken when changes are made to the meta-model and cannot be automatically resolved, therefore human
intervention is required.

Figure 1. The Co-Evolution Problem

Non-breaking changes consist of additions of new meta-model elements in a meta-model MM, which result in MM' without compromising MM conforming models, which in turn conform to MM'. For example, take the addition of a new element called Facility to meta-model MM depicted in Figure 1. After such a modification to MM, models conforming to MM still conform to MM', thus no co-evolution is necessary.

The above does not hold true always since general changes break models, even though automatic resolution can sometimes be performed in case of breaking and resolvable changes. For instance, meta-model MM had an element called Device with two sub-elements: PC and Mainframe. The meta-model change consisted of flattening the element hierarchy, meaning eliminating the Device element and the relation with its sub-elements, i.e., PC and Mainframe. Such a modification breaks the models that conform to MM since according to the new meta-model MM', the Device element can no longer be related. However, adding to each model element that is an instance of PC or Mainframe the shared properties of the Device element can automatically solve such breaking change, thus becoming compliant with MM'.

Regarding the solving of breaking and unresolvable changes, manual intervention is mandatory. This is so because certain changes over the meta-model require the introduction of additional information into the conformant models, the reorganization of the information already present, or even the deletion of some parts that cannot be automatically inferred. Take as an example the addition of a new non-null attribute to a particular element of MM' which was not previously specified in MM. The models cannot co-evolve automatically since the modeler or architect (the person that owns, builds, understands, and uses the models and its information) must take proper decisions regarding this specific change. In this example, either introduce the missing information related to the attribute or otherwise define default values.

Related Work

Existing approaches that solve this problem focus on two strategies: identifying the differences between the baseline and the target meta-models and applying a set of transformations on the model in order to be conformant to the new meta-model (Antonio Cicchetti et al. 2008b). The first approach uses a declarative evolution specification to define a difference meta-model which can be calculated from identified changes in the meta-model (Antonio Cicchetti et al. 2008b) whereas the second approach specifies meta-model evolution and model co-evolution through a sequence of operations in which each operation is then applied on meta-model and model level (Antonio Cicchetti et al. 2008a, Herrmannsdoerfer et al. 2009).

Our proposal aims at solving the co-evolution problem in the EA context by giving focus to the second strategy. Hence, we aim at compiling into a catalog a set of operations, categorized as breaking and resolvable changes, which can be used in the EA context to automatically modify the existing models, thus achieving meta-model conformance. Other approaches have addressed the co-evolution problem by proposing different frameworks for model transformation.

One approach to model co-evolution classifies the changes in atomic changes and define the process of co-evolution (Antonio Cicchetti et al. 2008b). Then, creates a differential meta-model with the identified changes, and it is classified in two new meta-models, the ones that are breaking and resolvable and the ones that are breaking and unresolvable. If there are no relations between the two meta-models, each one is executed independently. If relations between the two meta-models do exist, the co-evolution is done stepwise using user intervention.

COPE is a language used to satisfy two requirements: 1) reuse of recurring migration knowledge and 2) expressiveness to support domain-specific migrations. COPE allows creating coupled transactions that are
a combination of meta-model adaptation and model migration (Herrmannsdorfer et al. 2009). COPE solves the co-evolution problem by executing those coupled transactions and its execution does not require user intervention.

Another approach addressed the co-evolution problem by applying a set of automatic transformations thus solving the problem in one of three categories: addition, delete or rename (Gruschko et al. 2007). When a breaking and unresolvable change is found, the user should specify the way that the elements are going to change by creating a set of transformation rules using ETL.

In the EA context a platform capable of addressing breaking and unresolvable changes in model co-evolution was proposed that is based on two hypothesis: i) a meta-modeler knowing the rationale behind meta-model changes and capable of providing guidelines for model co-evolution, ii) the modeler being the one taking the final decisions about his models (Florez et al. 2012). The proposed languages for meta-modelers allows to specify changes in the meta-models and to propose corresponding changes in the models, which are then executed by an engine that automatically solves the changes in models that can be automatically solved.

**Research Proposal**

The Enterprise Architecture Model Co-Evolution Catalog (CO-EVOC) is a catalog of operations from the field of model-driven engineering adapted and applied to the EA field with the purpose of automating the evolution of EA models when changes to the EA modeling language, i.e., the EA meta-model occur.

The foundation of our research contribution is grounded on the specificities of the ArchiMate modeling language. The goal of ArchiMate is to provide an EA meta-model that fully enables integrated, cross-domain, enterprise modeling (Lankhorst 2004). For that purpose, ArchiMate serves as a domain-independent ontology for ensuring level coherence and supporting model and cross-model verification, which in turn concerns the evaluation of the conformance of EA models to the rules specified in the respective EA meta-models (Antunes et al. 2013).

As ArchiMate (The Open Group 2013) states, “a key challenge in the development of a general meta-model for enterprise architecture is to strike a balance between the specificity of languages for individual architecture domains, and a very general set of architecture concepts, which reflects a view of systems as a mere set of inter-related entities”. ArchiMate took the effort of minimizing the number of design restrictions so that the language could be used for most EA modeling tasks. As such, ArchiMate modeling the proverbial 80% of practical cases has been limited to the concepts that suffice for (The Open Group 2013).

**ArchiMate Core Concepts**

ArchiMate’s generic meta-model, illustrated in Figure 2, is built upon three core concepts: active elements (that perform), behavior elements (that describes the behavior of active elements) and passive elements (that hold the information implied in the behavior). Active and Behavior concepts have and hidden (internal) and exposed (external) counterparts.

**Definition 1.** “An active structure element is defined as an entity that is capable of performing behavior” (The Open Group 2013).

**Actors** at the business level, **Applications**, at the system level, and **Devices** at the technological level are examples of active elements. They are assigned to behavioral concepts that define the behavior to be performed.

**Definition 2.** “A behavior element is defined as a unit of activity performed by one or more active structure elements” (The Open Group 2013).

**Definition 3.** “A passive structure element is defined as an object on which behavior is performed” (The Open Group 2013).

Passive structure elements are usually information or data objects, but they may also be used to represent physical concepts. ArchiMate also distinguishes between external an internal view of systems. When looking at the behavioral aspect, these views reflect the principles of service orientation.
**Definition 4.** "A service is defined as a unit of functionality that a system exposes to its environment while hiding internal operations, which provides a certain value (monetary or otherwise)" (The Open Group 2013).

**Figure 2. Generic Meta-model: The Core Concepts of ArchiMate (adapted from (The Open Group 2013))**

External users, only perceive the service’s exposed functionality and value, together with non-functional aspects such as the quality of service, costs, etc. Services are accessible through interfaces, which constitute the external view on the active structural aspect.

**Definition 5.** “An interface is defined as a point of access where one or more services are made available to the environment” (The Open Group 2013).

An interface provides an external view of the service provider while hiding its internal structure.

**EA Model Co-Evolution Operations**

The specification of each EA model co-evolution operation relies on three fundamental aspects:

- **ArchiMate generic meta-model** (The Open Group 2013) (Figure 2);
- **Breaking and resolvable changes** from (Antonio Cicchetti et al. 2008a, 2008b) and
- **Heuristic rules** from (De Boer et al. 2005).

Take the example of ArchiMate version 3.0, that has renamed all the elements in the Technology Layer from `infrastructure <x>` to `technology <x>`. All Technology Layer elements defined as `Active Elements`, `Passive Structure Elements`, `Function Elements`, `Service Elements`, or `Interface Elements` would be subject to the `breaking and resolvable operation – rename meta-element` – which in turn would co-evolve the existing EA models accordingly to ensure meta-model conformance.

However, execution of these operations does not solve the co-evolution problem entirely in the context of EA. By applying such operations directly, the semantic integrity of the EA models may be compromised. This is caused by the ripple effect of performing a specific change that may violate the semantics of the relation connecting two or more EA model elements (De Boer et al. 2005). The ripple effect of a meta-model change into the model is the core subject of change’s impact analysis. The goal of such analysis is to take preemptive actions by observing the chain of events that might occur once a change to the model happens before the change really takes place (De Boer et al. 2005). This information can then be used to help in making a decision regarding the application of a change. Therefore, before committing a co-evolution operation, one must verify the model integrity every time a change to the meta-model occurs.

Thus, two aspects needed to be taken into consideration to determine the impact of a meta-model change in the models: (i) the type of change and (ii) the semantics of the relation connecting two or more elements. The type of change is given by the operations identified in (Antonio Cicchetti et al. 2008a, 2008b) whereas the relation semantics were compiled into a set of heuristic rules defined in (De Boer et al. 2005) and presented in Table 1. Taking both factors into consideration, a meta-model change can be applied to existing models if and only if the semantic integrity is preserved after executing the implied operations. For example, deleting a passive structure element that is accessed by a behavior element implies human intervention to
deal with dangling reference left in the behavior element (service or function). If the result of deleting or modifying a core concept has no impact whatsoever on the related elements, then the corresponding co-evolution operation can be automated and directly applied to the EA model elements. Note that re-factoring co-evolution operations that cause no impact to the structure and semantics of the elements, such as rename meta-element, can also be directly applied to the EA model elements.

<table>
<thead>
<tr>
<th>Relation X</th>
<th>Concept A &lt;X&gt; Concept B</th>
</tr>
</thead>
<tbody>
<tr>
<td>accesses</td>
<td>Delete (A) -&gt; NOTHING</td>
</tr>
<tr>
<td></td>
<td>Modify (A) -&gt; Modify (B)</td>
</tr>
<tr>
<td></td>
<td>Delete (B) -&gt; Dangle (A)</td>
</tr>
<tr>
<td></td>
<td>Modify (B) -&gt; Modify (A)</td>
</tr>
<tr>
<td>assignedTo</td>
<td>Delete (A) -&gt; Dangle (B)</td>
</tr>
<tr>
<td></td>
<td>Modify (A) -&gt; Dangle (B)</td>
</tr>
<tr>
<td></td>
<td>Delete (B) -&gt; NOTHING</td>
</tr>
<tr>
<td></td>
<td>Modify (B) -&gt; Modify (A)</td>
</tr>
<tr>
<td>usedBy</td>
<td>Delete (A) -&gt; Dangle (B)</td>
</tr>
<tr>
<td></td>
<td>Modify (A) -&gt; Modify (B)</td>
</tr>
<tr>
<td></td>
<td>Delete (B) -&gt; NOTHING</td>
</tr>
<tr>
<td></td>
<td>Modify (B) -&gt; Modify (A)</td>
</tr>
</tbody>
</table>

Table 1. Heuristic Rules Capturing Change in EA (De Boer et al. 2005)

Table 2 presents CO-EVOC (column 2) and the correlation between each CO-EVOC operation and its dual model-driven engineering operation.

<table>
<thead>
<tr>
<th>Model-Driven Engineering Co-Evolution Operations (Antonio Cicchetti et al. 2008a, 2008b)</th>
<th>EA Model Co-Evolution Operations Catalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminate meta-class</td>
<td>Eliminate Active Element</td>
</tr>
<tr>
<td></td>
<td>Eliminate Function Element</td>
</tr>
<tr>
<td></td>
<td>Eliminate Passive Structure Element</td>
</tr>
<tr>
<td></td>
<td>Eliminate Service</td>
</tr>
<tr>
<td></td>
<td>Eliminate Interface</td>
</tr>
<tr>
<td>Flatten Hierarchy</td>
<td>Flatten Active Element Hierarchy</td>
</tr>
<tr>
<td></td>
<td>Flatten Function Element Hierarchy</td>
</tr>
<tr>
<td></td>
<td>Flatten Passive Structure Element Hierarchy</td>
</tr>
<tr>
<td></td>
<td>Flatten Service Hierarchy</td>
</tr>
<tr>
<td></td>
<td>Flatten Interface Hierarchy</td>
</tr>
<tr>
<td>Rename meta-element</td>
<td>Rename Active Element</td>
</tr>
<tr>
<td></td>
<td>Rename Function Element</td>
</tr>
<tr>
<td></td>
<td>Rename Passive Structure Element</td>
</tr>
<tr>
<td></td>
<td>Rename Service</td>
</tr>
<tr>
<td></td>
<td>Rename Interface</td>
</tr>
<tr>
<td>Extract meta-class</td>
<td>Extract Active Element</td>
</tr>
<tr>
<td></td>
<td>Extract Function Element</td>
</tr>
<tr>
<td></td>
<td>Extract Passive Structure Element</td>
</tr>
<tr>
<td></td>
<td>Extract Service</td>
</tr>
<tr>
<td></td>
<td>Extract Interface</td>
</tr>
<tr>
<td>Inline meta-class</td>
<td>Inline Active Element</td>
</tr>
<tr>
<td></td>
<td>Inline Function Element</td>
</tr>
<tr>
<td></td>
<td>Inline Passive Structure Element</td>
</tr>
<tr>
<td></td>
<td>Inline Service</td>
</tr>
<tr>
<td></td>
<td>Inline Interface</td>
</tr>
<tr>
<td>Eliminate meta-property</td>
<td>Eliminate Relation</td>
</tr>
<tr>
<td>Move meta-property</td>
<td>Move Relation</td>
</tr>
<tr>
<td>Push meta-property</td>
<td>Push Relation</td>
</tr>
</tbody>
</table>

Table 2. Correlation between model-driven operations and CO-EVOC operations

The Flatten Hierarchy co-evolution, in the context of model-driven engineering, consists of deleting the element from which its sub-elements were specialized. In the context of EA, for example, the flatten-passive-structure-element-hierarchy(el) operation deletes a passive structure element and all the specialization relations connecting it to its sub-elements. The renaming of EA elements — rename-
active-element(el, new) – is a direct application of the rename meta-element operation. The extract operation – extract-function-element(ell, el2, rel) – involves creating a new element from an existing one and relating both elements by means of one of the relationships presented in Table 1, whereas the inline operation – inline-function-element(ell, el2, rel) – has an opposite behavior, i.e., deleting an element connected to another by one of Table 1 relations. A meta-property in this scope is reduced to the relationships between elements. Move Relation moves a relation from an element $e_1$ to an element $e_2$ – move-relation(ell, el2, rel) – in a straightforward manner without loss of information. Push Relation pushes a relation of a super-element and then clones in all the sub-elements – push-relation(rel, el).

Each proposed CO-EVOC operation denotes an algorithm that implements the said operation. For example, to implement the eliminate-function-element operation, we propose the following algorithm.

**Algorithm 1. eliminate-function-element(el)**

```plaintext
for all relations x in el do
  if x is accesses then
    eliminate-meta-class(el)
    continue
  end if
  if x is realizes then
    services = el.getRelatedElementsByRelationType(x)
    for all s in services do
      if s.getRelatedElementsByRelationType(realizedBy).length = 1 then
        eliminate-service(s)
      else
        continue
      end if
    end for
    eliminate-meta-class(el)
    continue
  end if
  if x is composedOf then
    for all c in el.getRelatedElementsByRelationType(x) do
      eliminate-function-element(c)
    end for
    eliminate-meta-class(el)
    continue
  end if
  markAsUnresolvable(el)
end for
```

For all the elements’ el relations, we verify the type of relation and heuristic rule (Table 1) that matches the operation’s nature, i.e. destructive or refactoring. If no impact whatsoever (NOTHING) then execute the dual co-evolution operation (eliminate-function-element <-> eliminate-meta-class); otherwise if Delete(B) (Table 1), we call eliminate-x-element(B) before executing the corresponding co-evolution operation on el. If el shares other relation types with other elements, the computation must be halted (by calling the markAsUnresolvable function) and human decision required before computation continues.

**Demonstration**

Throughout our consulting experience in EA projects in both public and private organizations, we have noted that organizations choose to start EA initiatives with a very simple and straightforward meta-model to facilitating their dissemination and understanding among the various stakeholders of the EA initiative. Over time, in the more advanced stages of the EA initiative, organizations tend to enrich the meta-model to
capture more complex situations. We present two common cases of meta-model enrichment. In the first case, the initial meta-model is the introduction of the Application Function and Application Interface types. This yields three changes as presented in table 3.

<table>
<thead>
<tr>
<th>#</th>
<th>Initial Meta-model</th>
<th>Enriched Meta-Model</th>
<th>CO-EVOC Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Data Objects are directly accessed by Application Components.</td>
<td>Data Objects are accessed only by Application Functions.</td>
<td>for all ac instance of AC do extract-function-element (af,ac,assignedTo) move-relation (ac,af,accesses) end for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MM-OP: extract-function-element (AF,AC,ASSIGNEDTO) move-relation (AC,AF,ACCESSSES)</td>
</tr>
<tr>
<td>2</td>
<td>Application Components directly realize Application Services.</td>
<td>Application Services realize only Application Functions.</td>
<td>for all ac instance of AC do extract-function-element (af,ac,assignedTo) move-relation (ac,af,realizes) end for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MM-OP: extract-function-element (AF,AC,ASSIGNEDTO) move-relation (AC,AF,REALIZES)</td>
</tr>
<tr>
<td>3</td>
<td>Application Components are directly associated to Application Services.</td>
<td>Application Components use Application Interfaces associated with Application Services.</td>
<td>for all ac instance of AC do extract-interface (ac,uses) move-relation (ac,ai,association) end for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MM-OP: extract-interface (AC,USES) move-relation (AC,Ai,ASSOCIATION)</td>
</tr>
</tbody>
</table>

Table 3. CO-EVOC Application Scenario - 1

The second case of meta-model enrichment relates to the categorization of the Application Component type into different types and configurations, as described in Table 4.

<table>
<thead>
<tr>
<th>#</th>
<th>Initial Meta-model</th>
<th>Meta-Model Evolution</th>
<th>CO-EVOC Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Application Component includes various types of components (Databases, Middleware, etc.).</td>
<td>Separation of Application Component into different types: Hierarchy creation.</td>
<td>for all t instance of T do extract-active-element (t,ac,specialization) end for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MM-OP: extract-active-element (T,AC,specialization)</td>
</tr>
<tr>
<td>2</td>
<td>Application Component incorporates both configuration and technology related aspects. For example, the same Application Component represents an acquired CRM application (technology) and its configuration for a given enterprise.</td>
<td>Separation of the Application Component type into Application Configuration and Application Component. What is acquired is now represented by the Application Component type, whereas configuration related aspects are expressed by the Application Configuration type, which can also be represented by a Data Object.</td>
<td>for all conf instance of CONF do extract-passive-structure-element (conf,ac,access) end for</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MM-OP: extract-passive-structure-element (CONF,AC,access)</td>
</tr>
</tbody>
</table>

Table 4. CO-EVOC Application Scenario - 2

Given that meta-model enrichment commonly occurs at secondary stages of EA initiatives, the EA repository often contains tens of thousands of objects, and therefore, the proposed meta-model evolutions imply a significant effort to transform the existing data. With our approach, meta-model evolution can be automated using the CO-EVOC operations indicated in Table 3 and Table 4. If no semantic violation is detected, the automation is complete. Otherwise, human intervention is required to decide between the various possible outcomes, and subsequent execution is automatic. Thus, human intervention is only necessary for decision making and not in actual operation execution.

A preliminary analysis done to a public health organization involving more than 30 hospitals points to fully automated operations in the order of 60% while the remaining 40% required a human decision. Nonetheless, there is still insufficient data at the moment to correlate the same results to a representative sample of EA initiatives.
Conclusion

In this paper, we presented CO-EVOC, an EA model co-evolution operations catalog to automate EA model co-evolution when changes to the EA meta-model occur. Each proposed operation is a combination of a model-driven operation-based approach and integrity heuristic rules, designed to assure the integrity of ArchiMate models. The goal was to provide a fully automatic approach to support EA model transformation, hence the proposed catalog of EA-specific co-evolution operations needed to be information preserving, i.e., capable of automatically transforming the existing models to become compliant with the newest version of the meta-model. We then applied our approach within a governmental organization.

The systematization of meta-model changes is by itself an important contribution because it enables the planning and the definition of common actions by transformation type, that ought to be applied to the EA model data stored in the EA repository. In addition, the automation of co-evolution operations yields practical gains, both in the necessary modeling effort as well as in the avoidance of modeling inconsistencies due to human error.

The identified limitations of this research at its current stage are twofold: the work of (De Boer et al. 2005) only provides a subset of ArchiMate relationships up to date. Even though the ArchiMate generic meta-model does not express other relationships, the assumption that those relationships ensure completeness of the heuristic rules may be flawed since ArchiMate models use other relationships besides the ones identified in the generic meta-model, hence possibly impacting related elements that are connected by such relationships. Secondly, no productivity assessment was yet performed to quantify effort gains regarding EA model transformation, before and after using the proposed CO-EVOC operations. Although preliminary data analysis points to 60% of changes fully automated, we have yet no sound evidence that the identified meta-model changes are representative of most EA initiatives.

Future efforts will encompass an assessment of the remaining ArchiMate relationships (aggregation, flow, specialization) with focus on the completeness of the heuristic rules used in our approach. Finally, the preliminary results will be consolidated through a larger sample of data to corroborate the percentage of completely automated changes and changes that require human intervention. These results will then be used to assess and quantify effort gains ex-ante and ex-post CO-EVOC implementation and demonstration.

Acknowledgments

The authors wish to thank the AMCIS reviewers for their valuable remarks. This research was supported by the Link Consulting’s project IT-Atlas (nº 11419, under the IAPMEI, 2020 Portuguese PO CI Operational Program).

REFERENCES


