LM2F: A LIFE-CYCLE MODEL MAINTENANCE FRAMEWORK FOR CO-EVOLVING ENTERPRISE ARCHITECTURE META-MODELS AND MODELS

Nuno Silva
Instituto Superior Técnico, University of Lisbon, 9nmcs0@gmail.com

Paloma Gonçalves
Instituto Superior Técnico, University of Lisbon, paloma.goncalves@tecnico.ulisboa.pt

Inês Leite
Instituto Superior Técnico, University of Lisbon, ines.leite@tecnico.ulisboa.pt

Pedro Sousa
Instituto Superior Técnico, University of Lisbon, pedro.sousa@linkconsulting.com

Miguel Mira da Silva
Instituto Superior Técnico, University of Lisbon, miguel.miradasilva@gmail.com

Follow this and additional works at: https://aisel.aisnet.org/ecis2019_rp

Recommended Citation
https://aisel.aisnet.org/ecis2019_rp/6

This material is brought to you by the ECIS 2019 Proceedings at AIS Electronic Library (AISeL). It has been accepted for inclusion in Research Papers by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Abstract

Enterprise Architecture (EA) models are tools that capture the entities and relationships that together describe the different enterprise domains. An EA meta-model defines a set of constructs and rules that explicitly specify how to build an EA model. Over time, to keep up with the evolutionary pressure due to internal and external factors, organisations need to evolve their EA meta-model and, consequently, update the existing EA model. In this paper, we present LM2F, a framework for co-evolving the EA model driven by a set of changes to the EA meta-model. The framework has two building blocks: a life-cycle temporal pattern based on the principles of life-cycle modelling and a catalogue of operators that update the EA model automatically. The LM2F goal is to reduce the modelling time required to manually update the EA model when the EA meta-model changes and to mitigate the error-proneness, also associated with manual modelling. LM2F also supports temporal analysis of the organisation’s EA due to its life-cycle properties. For illustrating the feasibility of LM2F, we describe a real use case scenario from the energy industry. The use case is realised using an implementation of LM2F in an EA management tool.

Keywords: Enterprise architecture, Meta-model, Model, Co-evolution, Life-cycle.
1 Introduction

Models are useful tools for not only capturing the entities and relationships within a given domain but also helping to reduce and master the complexity of the related system design, thus improving stakeholder communication (Proper, Verrijn-Stuart, and Hoppenbrouwers, 2005). In this regard, an Enterprise Architecture (EA) model expresses and disseminates the enterprise’s organisational structure, business processes, information systems, and IT infrastructure (Lankhorst, 2009). When taking into account enterprise transformation and the role of EA towards the achievement of the enterprise’s goals and the business/IT alignment, one must consider the evolution of EA over time, and consequently, the evolution of the EA model.

The challenges associated with the evolution and maintenance of the EA model are documented in the literature (Farwick, Breu, et al., 2013; Fill, 2018; Kaisler, Armour, and Valivullah, 2005; Lucke, Krell, and Lechner, 2010; Rouhani et al., 2015). The common practice in updating the EA model resumes to a collection of EA information via stakeholder interviews which are then manually entered into an enterprise architecture management (EAM) tool (Farwick, Agreiter, et al., 2011). As stated by Farwick et al. this is a time-consuming manual task, that can only be executed by EA specialists, besides being error-prone when the complexity of changes rises (Farwick, Breu, et al., 2013). Enterprises, to support the EA evolutionary aspect, have EAM tools that follow the principles of life-cycle modelling (Buckl, Ernst, et al., 2009; Buckl, Matthes, et al., 2011; Sousa et al., 2009; The Open Group, 2018). Life-cycle modelling allows for temporal analysis of the EA model which becomes relevant when assessing future scenarios, in which a given EA model may have several ramifications with different degrees of dependencies and complexities.

The principles that guide life-cycle modelling as an approach used in the maintenance of the EA model should also be considered when changes to the EA meta-model (i.e., an explicit description - defining constructs and rules - of how to build an EA model), occur. Since the definition of the EA meta-model happens at the early stages of EA initiatives, the EA meta-model is likely to evolve in subsequent stages. Causes that force an EA meta-model to evolve can be (1) internal - when the needs of expressiveness increase along with the evolution and scope of the EA initiatives; and (2) external - when the standards or compliance rules change (N. Silva, Mira da Silva, and Sousa, 2018b). Moreover, the iterative nature of the meta-model and model construction process also poses an evolutionary pressure onto the EA meta-model and model (Florez et al., 2012).

The challenge of evolving the EA meta-model has to do with the consequent co-evolution of the EA model that may no longer conform (i.e., built on the entity types and relationship types encoded in the meta-model (Cicchetti et al., 2008a)) to the new version of the EA meta-model. The need to co-evolve the EA model combined with its inherent complexity puts a strenuous effort in enterprises that seek to maintain their existing EA, leading ultimately to an ineffective EAM.

When taking into account the spectrum of EA meta-model changes, one notices that not all the changes compromise the EA model conformance. For instance, the addition of new entity types typically causes no structural changes on the existing EA model; although, a revision of existing entity types and their respective behaviour may be required, thus potentially leading to other EA meta-model changes that might have a structural impact on the EA model. However, the deletion and re-factoring of entity types and relationship types in the EA meta-model, which typically happens when organisations need to increase the modelling capacity and structural expressiveness of their EA model, may put the model’s conformance at risk. These types of changes are usually responsible for most of the modelling effort in the evolution of the meta-model since they also force changes to the existing model to achieve model conformance (Cicchetti et al., 2008a).

Existing approaches in the field of model-driven engineering (MDE) have addressed the co-evolution challenge (Anguel, Amirat, and Bounour, 2015; Hebig, Khelladi, and Bendraou, 2017; Paige, Matragkas, and Rose, 2016; Rose et al., 2009). However, as far as our understanding of the literature on co-evolution goes, it appears that such techniques have yet to tackle the co-evolution challenge, from an EA perspective,
while taking into account the specifics of life-cycle modelling (Buckl, Matthes, et al., 2011; Sousa et al., 2009; The Open Group, 2018). Hence, the research question is: How can we combine both life-cycle modelling and MDE co-evolution approaches to address the co-evolution challenge in EA? The Life-Cycle Model Maintenance Framework (LM2F) is a framework capable of (semi-)automating the co-evolution process while preserving a history of the different EA states. The LM2F aims to achieve the following objectives:

1. Reduce the amount of time required to manually update the EA model when the EA meta-model changes; and
2. Mitigate the error-proneness, compared to the traditional manual approach.

We based our research work in the Design Science Research Methodology (DSRM) (Peffers et al., 2007). DSRM helps to structure and conduct our research by applying a well-defined set of steps. To demonstrate the framework’s feasibility, we applied an implementation of LM2F to one of the largest Portuguese organisations in the energy industry. Results proved that LM2F could reduce the time of co-evolving the EA meta-model and EA model and avoid model inconsistencies due to human error. Furthermore, LM2F supports temporal analysis by providing temporal snapshots of the EA meta-model and the EA model. Finally, we would like the reader to take into consideration the following assumptions throughout the rest of the paper. We refer to the term co-evolution of the EA meta-model and EA model as a set of structural changes or transformations that can affect the structure of the respective EA model. Therefore, we do not discuss the potential impacts of changing the semantics and behaviour of each EA meta-model entity type or relationship type. Hence, EA model conformance refers only to the structural level, thus ignoring the underlying semantics of each concept.

The remainder of the paper is as follows. Section 2 presents the theoretical foundations on which we built LM2F. In Section 3 we describe LM2F and the building blocks comprising it. Sections 4 presents the results of applying an implementation of LM2F to an organisation’s EA. Section 5 enumerates related work on the subject and, finally, Section 6 concludes the paper and points out future research efforts.

2 Background

The following sections describe the theory behind model co-evolution and life-cycle modelling, which we used in the design of LM2F.

2.1 Meta-Model and Model Co-Evolution

Co-evolution is a process that is required when an element (entity type or relationship type) from the meta-model changes and the model no longer conforms to the new meta-model. After performing an evolution $\Delta$ 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 $\Delta$ (Cicchetti et al., 2008a; Florez et al., 2012). This type of co-evolution strategy is called Predefined Resolution strategy, in which an approach enables automation by applying predefined resolution strategies, where possible (Hebig, Khelladi, and Bendraou, 2017).

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, subtracting, and refactoring. Therefore, changes that occur on a meta-model may have different effects on the related models. These changes are classified as follows (Cicchetti et al., 2008a,b):

- **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.

Non-breaking changes consist of additions of new meta-model elements in a meta-model MM, which result in MM’ without compromising MM conforming model, which in turn conforms to MM’. The above does not always hold since most changes of the ∆ spectrum break the models, even though automatic resolution can be performed in case of breaking and resolvable changes. Regarding the solving of breaking and unresolvable changes, manual intervention is mandatory. The necessity of manual intervention has to do with specific changes over the meta-model that require the introduction of additional information into the conforming models, the reorganisation of the information already present, or even the deletion of some parts that cannot be inferred automatically.

### 2.2 Life-Cycle Modelling

The work from (Buckl, Matthes, et al., 2011) describes three modelling techniques for enterprise transformation, one of those being *life-cycle modelling*. A life-cycle indicates that a single element of the EA evolves going through different states (Sousa et al., 2009; Tribolet, Sousa, and Caetano, 2014; Xavier, Vasconcelos, and Sousa, 2017). Although customisation is possible to fit the organisation’s needs, an EA element’s life-cycle is fundamentally composed of five invariant states:

- **Conceived.** The state in which an EA element is planned but its materialisation into a productive element did not yet start.
- **Gestating.** The state describes an EA element being constructed or acquired to become productive.
- **Alive.** The state assigned to an EA element the moment it becomes productive and starts actively playing purposeful roles and creating value to the organisation. Taking into account only the five invariant states, we consider in this paper the Alive state as the only productive state.
- **Retired.** The state is triggered when an Alive EA element no longer plays an active role in the organisation’s transactions and processes to create value. A transition from the Gestating state to the Retired state can happen due to both internal and external factors that may invalidate the moving of a set of gestating elements into production.
- **Removed.** The state represents the post-Retired state in which the EA element no longer has an impact on the remaining elements. A removed element is unable to interact with Alive elements. An EA element can move from Conceived state directly to a Removed state if it never entered in a Gestating state, meaning that it never went beyond an idea.

According to Buckl et al. the most basic way of modelling temporal aspects of EA elements is to assign validity periods for each element individually (Buckl, Matthes, et al., 2011). Xavier et al. define such periods as Begin Date” (BD) and "End Date” (ED) which are attributes of each EA element (Xavier, Vasconcelos, and Sousa, 2017). With such attributes, it is possible to determine a set of restrictions concerning the validity of any state. Therefore, a state is only valid during the period specified by its BD and ED, respectively.

However, this set of validity restrictions affect the relationships among the entities. Buckl et al. argue that these restrictions are not mandatory for all types of relationships (Buckl, Matthes, et al., 2011). So, to provide a distinction between required and non-required restrictions, Buckl et al. describe two types of relationships for their temporal qualities (Buckl, Matthes, et al., 2011):

- **Synchronic Relationship.** Consists of the relationships that are valid only among entities whose validity periods intersect.
- **Diachronic Relationship.** Consists of the relationships that are valid regardless of the validity periods of the entities; thus the issue of the intersection of the periods is not taken into account.
Xavier et al. further elaborate on this notion of synchronic and diachronic relationships and define a set of temporal rules, as shown in Table 1, which allows both the verification and correction of inconsistencies in the relationships among EA entities to the extent that they will evolve (i.e., change their current state). They call temporal rules since they define, regarding the EA entities, states through its temporal attributes that can be used to verify or correct inconsistencies in the relationship between EA entities.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Rule Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Rule</td>
<td>The composition relationship is valid only if the entities in them are involved (&quot;whole&quot; and the &quot;parts&quot;) are in the &quot;alive&quot; state in the same time period.</td>
</tr>
<tr>
<td>Aggregation Rule</td>
<td>In aggregation relationship the &quot;whole&quot; entity enters to the state &quot;alive&quot; when all the entities &quot;parts&quot; are in &quot;alive&quot; state.</td>
</tr>
<tr>
<td>Create Rule</td>
<td>The create relationship, the created entity should only go to the &quot;alive&quot; state as creative entity is in &quot;alive&quot; state.</td>
</tr>
<tr>
<td>Synchronic rule for Relationship</td>
<td>In the relationship among A and B where the A acts on B and B depends on A so B should only go to the &quot;alive&quot; state and keep in this state while A is &quot;alive&quot;.</td>
</tr>
<tr>
<td>Discontinuation Rule</td>
<td>The new entities added to EA cannot relate to entities in a state of discontinuation.</td>
</tr>
</tbody>
</table>

Table 1: A general definition of the temporal rules - from (Xavier, Vasconcelos, and Sousa, 2017).

To define these rules, Xavier et al. consider the characteristics of each ArchiMate relationship. The discontinuation rule is derived from the insertion in the EA model of the retired state and is used to identify the moment in which an EA entity enters the retired state (Xavier, Vasconcelos, and Sousa, 2017). Xavier et al. temporal rules act on relationships between EA entities and are divided into Inconsistencies Verification Rules (IVRs) and Inconsistencies Correction Rules (ICRs). IVRs detect temporal inconsistencies while ICRs correct such temporal inconsistencies.

3 LM2F: The Life-Cycle Model Maintenance Framework

This section presents LM2F: a Life-Cycle Model Maintenance Framework for co-evolving EA meta-models and models. LM2F is composed of two building blocks. The first building block refers to the life-cycle temporal pattern (described in Section 3.1). The second building block specifies a catalogue of co-evolution operators used to co-evolve the EA meta-model and respective model (see Section 3.2).

3.1 The Life-cycle Temporal Pattern

The life-cycle temporal pattern presented in this section stands on the work from (Buckl, Matthes, et al., 2011; Sousa et al., 2009; Tribolet, Sousa, and Caetano, 2014). The purpose of such an artefact is twofold. First, to enable temporal analysis by saving the previous states of the entity types, relationship types, entities, and relationships comprising both the EA meta-model and EA model, respectively. Second, to provide the means to verify temporal inconsistencies through IVRs and correct them using ICRs. The life-cycle temporal pattern is formalised as a deterministic finite state machine ($\Sigma$, $S$, $s_0$, $\delta$, $F$), where:

- $\Sigma$ is the input alphabet (a finite, non-empty set of symbols), corresponding to a set of co-evolution operators.
- $S$ is a non-empty set of five invariant states: Conceived, Gestating, Alive, Retired, Removed.
- $s_0$ is the initial state and an element of $S$ (Conceived).
- $\delta$ is the state-transition function: $\delta: S \times \Sigma \rightarrow S$.
- $F$ is the final state and a state of $S$ (Removed).

Figure 1 illustrates the life-cycle temporal pattern. The following concepts define the pattern:
• **Life-cycle**: The life-cycle concept defines the interface with the entity types, relationship types, entities, and relationships that comprise both the EA meta-model and the EA model. The life-cycle keeps an instance of a particular state - its current state - and a list of particular states that defines the past states.

• **State**: Defines an interface that encapsulates the responsibilities associated with a concrete life-cycle state.

• **Conceived**: The conceived state implements the life-cycle behaviour corresponding to the state itself.

• **Gestating**: The gestating state implements the life-cycle behaviour corresponding to the state itself.

• **Alive**: The alive state implements the life-cycle behaviour corresponding to the state itself.

• **Retired**: The retired state implements the life-cycle behaviour corresponding to the state itself.

• **Removed**: The removed state implements the life-cycle behaviour corresponding to the state itself.

![Figure 1: The life-cycle temporal pattern.](image)

The life-cycle, being the primary interface with the entity types and relationship types of the EA meta-model and the entities and relationships of the EA model, each entity type, relationship type, entity, and relationship does not need to deal with the state objects directly. It is of the life-cycle responsibility to delegate a specific state from each of the five fundamental states. As referred to in Section 2.2, the defined states for each life-cycle can be customised according to the organisation’s specific needs. LM2F establishes the five fundamental sates as the life-cycle’s baseline, according to the existing literature on the subject.

### 3.2 The Co-Evolution Operators Catalogue

The co-evolution is described as the sum of changes done to the meta-model that in turn is propagated to the respective model. Such changes to the meta-model and conforming model are applied via a set of operators that according to (Wachsmuth, 2007) are grouped into three co-evolution categories:

• **Construction operators**: Operators that belong to this category enrich the domain and modelling expressiveness by introducing new meta-model entity types and relationship types.

• **Destruction operators**: Operators that belong to this category reduce the domain and modelling expressiveness by eliminating existing meta-model entity types and relationship types.

• **Re-factoring operators**: Operators that belong to this category neither reduce nor augment the domain and modelling expressiveness. This type of operators changes the structure, and possibly the semantics of the meta-model, without compromising modelling expressiveness power.
Algorithm 1: Introduce Relationship

Input: RelationshipType, SrcEntityType, DestEntityType, ConceivedDate

conceivedState ← Conceived(conceivedDate);
RelationshipType.getLifecycle().setState(conceivedState);

for se instanceof SrcEntityType do
    for de instanceof DestEntityType do
        r ← RelationshipType(se, de);
        r.getLifecycle().setState(conceivedState);
    end
end

In this section, we present LM2F’s second building block, a co-evolution operators catalogue that defines a set of construction, destruction, and re-factoring operators based on a predefined resolution strategy (Hebig, Khelladi, and Bendraou, 2017). The novelty of this approach is that each operator acts on the life-cycle state of both the entity types and relationship types present in the EA meta-model and the entities or relationships present in the EA model rather than acting on the elements themselves. For example, when applying the Eliminate Element operator, what the operator does is 1) adds the current life-cycle state of a given entity type to a list of past states, 2) marks the current life-cycle state as Retired and then 3) iterates over all entity type’s instances and applies the same life-cycle update logic as in the entity type.

Algorithm 2: Eliminate Relationship

Input: RelationshipType, RetiredDate

retiredState ← Retired(RetiredDate);

if checkTemporalInconsistencies(RelationshipType, RelationshipType.getTemporalType()) == false then
    correctTemporalInconsistency(RelationshipType, RelationshipType.getTemporalType());
end

currentState ← RelationshipType.getLifecycle().getState();
currentState.setEndDate(RetiredDate);
RelationshipType.getLifecycle().getPastStates().add(currentState);
RelationshipType.getLifecycle().setState(retiredState);

for r instanceof RelationshipType do
    if checkTemporalInconsistencies(r, r.getTemporalType()) == false then
        correctTemporalInconsistency(r, r.getTemporalType());
    end
    rCurrentState ← r.getLifecycle().getState();
    rCurrentState.setEndDate(RetiredDate);
    r.getLifecycle().getPastStates().add(rCurrentState);
    r.getLifecycle().setState(retiredState);
end

Each operator, belonging to a co-evolution category, can be classified into one of two types 1:

• Atomic operators. An operator that cannot be further decomposed. These operators are: Introduce Entity, Introduce Relationship, Eliminate Entity, and Eliminate Relationship.

• Complex operators. An operator composed of two or more atomic or complex operators. Such operators are: Rename Entity, Flatten Hierarchy, Extract Entity, Inline Entity, Merge Entities, Split

1 Due to paper length restrictions, we refer the reader to a more detailed description of each co-evolution operator in (N. Silva, Mira da Silva, and Sousa, 2018b; N. Silva, Sousa, and M. M. d. Silva, 2019).
Entity, Entity to Relationship, Relationship to Entity, Rename Relationship, and Move Relationship.

**Algorithm 3:** Check Temporal Inconsistencies

**Input:** RelationshipType, Type

**Parameters:** isValid ← false,

  srcEntityState ← RelationshipType.getSrcEntity().getLifecycle().getState(),
  destEntityState ← RelationshipType.getDestEntity().getLifecycle().getState(),
  srcBeginDate ← srcEntityState.getBeginDate(),
  srcEndDate ← srcEntityState.getEndDate(),
  destBeginDate ← destEntityState.getBeginDate(),
  destEndDate ← destEntityState.getEndDate()

**Output:** true or false

```plaintext
begin
  switch Type do
    case C do
      isValid ← ((srcBeginDate = destBeginDate AND srcEndDate = destEndDate) AND
      (destBeginDate = srcBeginDate AND destEndDate = srcEndDate));
    end
    case A do
      isValid ← ((srcBeginDate > max(destBeginDate_1, ..., destBeginDate_n) AND
      srcEndDate <= min(destEndDate_1, ..., destEndDate_n)) AND
      (destBeginDate <= srcBeginDate AND destEndDate >= srcEndDate));
    end
    case Cr do
      isValid ← (srcEndDate > destBeginDate AND destBeginDate <= srcEndDate);
    end
    case SR do
      isValid ← ((srcBeginDate <= destBeginDate AND srcEndDate >= destBeginDate) AND
      (destBeginDate >= srcBeginDate AND destEndDate <= srcBeginDate));
    end
    case D do
      isValid ← (srcElState instanceof Removed AND srcBeginDate < destBeginDate AND
      destBeginDate > srcBeginDate);
    end
    otherwise do
      continue;
  end
end
return isValid;
```

Algorithm 2 describes the Eliminate Relationship atomic operator, which receives as input an EA meta-model relationship type RelationshipType and a retire date RetiredDate.

The remainder of this section presents two specifications regarding two atomic operators - Introduce Relationship and Eliminate Relationship - as well as a specification of temporal rules for checking and correcting temporal inconsistencies throughout the co-evolution process. We refer the reader for more details regarding the atomic and complex operators in (N. Silva, Sousa, and M. M. d. Silva, 2019).

Algorithm 1 describes the Introduce Relationship atomic operator, which receives as input an EA meta-model relationship type RelationshipType, the EA meta-model's source entity type SrcEntityType and destination entity type DestEntityType of RelationshipType, and a conceived date ConceivedDate. First,
the Introduce Relationship operator creates a Conceived state with *ConceivedDate* as the state’s *beginDate*. Then, it sets the life-cycle’s current state of *RelationshipType* to the new Conceived state. Finally, for all EA model instances of *SrcEntityType* and *DestEntityType*, the operator creates a new EA model instance of *RelationshipType* and sets its life-cycle current state to the new Conceived state.

### Algorithm 4: Correct Temporal Inconsistency

**Input:** *RelationshipType, Type*

**Parameters:**

\[ \text{isValid} \leftarrow \text{false}, \]
\[ \text{srcEntityState} \leftarrow \text{RelationshipType}.\text{getSrcEntity().getLifecycle().getState()}, \]
\[ \text{destEntityState} \leftarrow \text{RelationshipType}.\text{getDestEntity().getLifecycle().getState()}, \]
\[ \text{srcBeginDate} \leftarrow \text{srcEntityState}.\text{getBeginDate()}, \]
\[ \text{srcEndDate} \leftarrow \text{srcEntityState}.\text{getEndDate()}, \]
\[ \text{destBeginDate} \leftarrow \text{destEntityState}.\text{getBeginDate()}, \]
\[ \text{destEndDate} \leftarrow \text{destEntityState}.\text{getEndDate()} \]

**begin**

\[ \text{switch Type do} \]
\[ \text{case C do} \]
\[ \quad \text{if srcBeginDate \neq destBeginDate then} \]
\[ \quad \quad \text{destBeginDate} \leftarrow \text{srcBeginDate}; \]
\[ \quad \text{else} \]
\[ \quad \quad \text{srcBeginDate} \leftarrow \text{destBeginDate}; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{case A do} \]
\[ \quad \text{if srcBeginDate < max(destBeginDate_1, ..., destBeginDate_n) OR} \]
\[ \quad \quad \text{srcBeginDate} \geq \max(destBeginDate_1, ..., destBeginDate_n) \text{ then} \]
\[ \quad \quad \text{srcBeginDate} \leftarrow \max(destBeginDate_1, ..., destBeginDate_n); \]
\[ \quad \text{else} \]
\[ \quad \quad \text{srcEndDate} \leftarrow \min(destEndDate_1, ..., destEndDate_n); \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{case Cr do} \]
\[ \quad \text{srcEndDate} \leftarrow \text{destBeginDate}; \]
\[ \text{end} \]
\[ \text{case SR do} \]
\[ \quad \text{if srcBeginDate} > \text{destBeginDate OR destBeginDate} < \text{srcBeginDate then} \]
\[ \quad \quad \text{destBeginDate} \leftarrow \text{srcBeginDate}; \]
\[ \quad \text{else} \]
\[ \quad \quad \text{destEndDate} \leftarrow \text{srcEndDate}; \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{case D do} \]
\[ \quad \text{warn(}\text{Relationship}); \]
\[ \text{end} \]
\[ \text{otherwise do} \]
\[ \quad \text{continue}; \]
\[ \text{end} \]
\[ \text{end} \]

**end**
First, the Eliminate Relationship operator creates a Retired state with RetiredDate as the state’s beginDate. Then, it checks for temporal inconsistencies at the meta-model level. If any temporal inconsistencies are found, we call the Correct Temporal Inconsistency procedure before resuming computation; otherwise, we (1) set the current state’s endDate to RetiredDate, (2) add the current state to the list of pastStates, and (3) set the life-cycle current state of RelationshipType to the new Retired state. Afterwards, we apply the same steps to all EA model relationships \( r \) instances of RelationshipType.

When considering both the co-evolution process and the possible existence of temporal inconsistencies in the relationship types among entity types, and consequently in the relationships among entities, one must take into account whether or not a specific IVR is being violated since a violation of an IVR will compromise the correctness of the EA meta-model and EA model during the co-evolution process. To address potential temporal inconsistencies, the Eliminate Relationship operator takes into consideration the temporal rules defined in (Xavier, Vasconcelos, and Sousa, 2017) to assess temporal inconsistencies of an EA meta-model change before changing the life-cycle state of the entity types, relationship types, entities, and relationships comprising both the EA meta-model and the EA model, respectively. Therefore, for each relationship type and respective instances, the corresponding IVR is computed. If no temporal inconsistencies are found, the operator proceeds with the computation by altering the life-cycle state.

Algorithm 3 specifies the validation logic of each IVR as defined in (Xavier, Vasconcelos, and Sousa, 2017). The procedure receives as input a relationship type RelationshipType and the respective temporal type Type. If the type corresponds to a synchronic type, the model remains consistent iff the SR-IVR formalisation holds. The same rationale applies to the remaining IVRs: C-IVR, A-IVR, Cr-IVR, and D-IVR. Algorithm 4 specifies the correction logic of each ICR as defined in (Xavier, Vasconcelos, and Sousa, 2017). The procedure receives as input a relationship RelationshipType and the respective temporal type Type. If the type corresponds to a synchronic, composition, aggregation, creation, or discontinuation type the respective ICR is performed.

## 4 Demonstration

In order to demonstrate LM2F in practice, we have selected a real scenario of a large Portuguese company in the energy industry. This scenario, named Demo Corp, has a meta-model which included, prior to the co-evolution process, 42 entity types and 51 relationship types. The model conforming to the Demo Corp meta-model had 1306 entities and 1419 relationships, also prior to the co-evolution process.

The goal of this evaluation is to demonstrate that 1) LM2F is useful for EA modellers since it can reduce the EA model maintenance time when changes to the EA meta-model occur, and 2) LM2F can mitigate error-proneness compared to a manual approach. To achieve this, we used two meta-model snapshots concerning two Demo Corp meta-model versions. The first snapshot, from 24-01-2018, had a total of 42 entity types and 51 relationship types. The second snapshot, from 16-08-2018, had a total of 55 entity types and 67 relationship types. We identified a total of 39 changes to the meta-model during this period, that yield 1676 changes in the EA model. In the remainder of this section, we describe the application of three co-evolution complex operators - Split Element, Rename Element, and Move Relationship - and respective results.

The Split Element operator was identified from comparing both meta-model snapshots where in the first a Card entity type exists whereas in the second three new entity types were identified: Card-Solution, Card-System, and Card-Component. In the scope of Demo Corp, a Card represents information regarding a given System, such as system’s properties, user information, and release information. This entity type was split into three new entity types as a means of capturing the information details regarding a given Solution or Component of a System. The Rename Element operator was identified in the same manner. In the meta-model version corresponding to the first snapshot we have a Business Role entity type, though in the second version of the meta-model the entity type was replaced for Position. Finally, the Move Relationship operator was identified due to the EA meta-model Participates relationship type being connected to the Business Actor and Business Process entity types respectively in the first snapshot of the...
meta-model, while on the second that same relationship type was connected to the Business Actor and the Information Flow entity types instead. Figure 2a and Figure 2b show the number of impacted model entities and relationships prior to and after the execution of the three operators.

Figure 2: Number of Demo Corp model entities and relationships marked for migration.

Figure 3: LM2F temporal analysis implemented in an EAM tool.

Figure 2a shows that regarding the Split Element operator a total of 291 EA model entities and 873 relationships were marked for migration. As for the Rename Element operator, 345 entities and 397 relationships were impacted. Finally, the Move Relationship operator marked for migration 167 relationships in the Demo Corp’s EA model. Figure 2b illustrates the number of EA model entities and relationships created and those that had their life-cycle states changed after the model migration phase of the co-evolution process. A total of 1164 entities and 1746 relationships were created or changed their life-cycle state after the execution of the Split Element operator. As for the Rename Element operator, 690 model entities and 794 relationships had their life-cycle state changed to Retired. Finally, results from applying the Move Relationship operator show a total of 334 relationships that were either created or had their life-cycle state changed to Retired.

Figure 3 illustrates a representation of LM2F’s life-cycle temporal pattern in a proprietary EAM tool. This particular case illustrates the properties and relationship types of the Demo Corp’s Business Actor entity type that changed during the period between 12-06-2018 and 08-08-2018, as a result of applying the Move Relationship operator. As observed, the Participates relationship type between the Business Actor and the Business Process entity types had its life-cycle state changed to Retired - represented by the red colour - and another Participates relationship type was created between the Business Actor and the Information Flow entity types with its life-cycle state set to Gestating - shown in green.
As a result of applying LM2F in a real co-evolution scenario, we could prove LM2F’s feasibility and applicability in 1) (semi-)automating the co-evolution process in an EA context, thus achieving objectives 1 and 2 (see Section 1); 2) enabling temporal analysis, as a result of changing the life-cycle state of both the EA meta-model and EA model elements; 3) addressing the co-evolution challenge in an EA context and applied to a real and complex scenario; and 4) ensuring model conformity and temporal consistency in all steps of the co-evolution process.

5 Related Work

The evolution and maintenance of EA models is a subject of research addressed in the EA body of knowledge due to the scarcity of mechanisms that ensure the actuality of the EA repository and the underlying complexities of maintaining the EA model (Farwick, Breu, et al., 2013; Lucke, Krell, and Lechner, 2010; Winter et al., 2010). The literature provides a set of approaches that deal with this problem, ranging from evolution-specific meta-models, ontology models, to heuristics and machine learning models (Boer et al., 2005; Buckl, Ernst, et al., 2009; Dam, Lê, and Ghose, 2016; Farwick, Pasquazzo, et al., 2012; Fill, 2018; Franke et al., 2009; Gaaloul and Guerreiro, 2015; Johnson, Ekstedt, and Lagerstrom, 2016; N. Silva, Mira da Silva, and Sousa, 2017a,b). Our focus, however, is on identifying the existing approaches that deal with the challenges of EA model evolution or maintenance from a co-evolution perspective.

The work from (Florez et al., 2012) presents a co-evolution approach for EA taking into consideration the following hypothesis: (1) the meta-model is built by one meta-modeller, and a conforming model built by one modeller, (2) the meta-model evolution takes place in the event that meta-modellers defining the changes over the meta-model add alternatives to solve the changes on the model that are not automatically resolvable. With their approach, a modeller can apply the co-evolution specification to a model that conforms to the changed meta-model by choosing among the available alternatives previously defined by the meta-modellers. This helps to avoid potential inconsistencies introduced by modeller decisions since they must follow specific rules established by the meta-modellers. The approach also enables a step-by-step execution thus allowing modellers to store the process, restarting it in a specific point, or even rolling back changes in case of modellers change decisions about the co-evolution of the model.

While (Florez et al., 2012) focus on handling breaking and non-resolvable changes (i.e., changes that cannot be automatically inferred and applied to the EA model without human intervention), LM2F addresses co-evolution from a breaking and resolvable change (i.e., changes that can be automatically inferred and applied to the EA model) standpoint. An advantage of using the approach presented in (Florez et al., 2012) is that modellers cannot perform illogical changes on the model to make it conforming to an evolved meta-model when breaking and non-resolvable changes occur. Hence, modellers must choose from a set of logical defined changes provided by meta-modellers. LM2F also provides the same advantages as (Florez et al., 2012) regarding breaking and resolvable changes since in order to automate the co-evolution process, modellers must perform a set of changes presented in LM2F’s co-evolution operators catalogue module, otherwise manual modelling effort is required.

Furthermore, (Florez et al., 2012) guarantees atomicity and consistency throughout the co-evolution process as the execution of the co-evolution is based on a set of atomic modifications to the meta-model, where each transition of the model happens after the correspondent transition on the meta-model. Consequently, model conformance is achieved as the co-evolution is taking place. Also, (Florez et al., 2012) argue that their approach not only improves the time and number of actions required by modellers in the co-evolution process but also ensures the model conformance after each change over the correspondent meta-model. Moreover, the co-evolution in EA models with a large number of instances is possible with a simple intervention by the modeller.

We would argue that the fundamental difference between LM2F and (Florez et al., 2012), besides addressing distinct types of meta-model changes, is the ability of LM2F to provide temporal analysis regarding both the EA meta-model and EA model. By combining the life-cycle modelling principles with the set of changes guiding co-evolution automation, one can reduce the overall manual effort and error-
proneness of maintaining both the EA meta-model and EA model while preserving different snapshots of the EA, each one corresponding to specific life-cycle states of the EA meta-model entity types and relationship types and EA model entities and relationships in a given point in time. Regarding tool support, a comparison of EAM vendor tools concerning meta-model flexibility and meta-model/model co-evolution identified some meta-model flexibility features but no support for meta-model and model co-evolution (N. Silva, Mira da Silva, and Sousa, 2018a).

6 Conclusion

In this paper, we presented LM2F, a life-cycle model maintenance framework for co-evolving the EA meta-model and conforming model. The presented artefact provides an implementable framework capable of reducing the EA model maintenance time due to meta-model changes while enabling temporal analysis regarding the EA meta-model and EA model. We applied an implementation of LM2F, to a real use case in collaboration with a Portuguese organisation from the energy industry, to assess the artefact’s feasibility and applicability in practice. Results suggest that the systematisation of meta-model changes is by itself a contribution since it enables the planning, through temporal model analysis, and the definition of collective actions by transformation type, which ought to be applied to the EA model data stored in the EA repository. Moreover, the implementation of LM2F yields practical gains, both in the required modelling time as well as in the avoidance of modelling inconsistencies due to human error.

The identified limitations of the research artefact are as follows. LM2F takes into account (Xavier, Vasconcelos, and Sousa, 2017) approach which by itself has limitations regarding the number of verified inconsistencies, hence limiting the validation of the EA meta-model. Also, LM2F is only capable of automating structural changes while using IVRs and ICRs to ensure temporal consistency between the meta-model entity types and model entities. Hence, LM2F does not take into account the behaviour associated with each element, meaning the intervention of both the meta-modeller and modeller is needed to assess semantic consistency. Automation is also limited to the existing catalogue of operators.

A rather significant limitation has to do with both the meta-model and model semantics which are not considered in the current version of LM2F. Semantics is a highly debated issue in the EA community as each stakeholder, despite talking a common language, sometimes can have a different understanding of the language. Therefore, it is difficult to compare semantic definitions automatically.

Finally, limitations regarding the demonstration of LM2F should be discussed. In particular, despite the dimension and EA maturity of the organisation, a single case study is not representative enough to claim solid conclusions regarding the effectiveness of a solution. Hence, the reader should consider these as preliminary results as more applications of LM2F must be made so that the results can be representative.

Future efforts encompass the extension of the co-evolution operators catalogue with new complex operators and the implementation of the remaining operators with posterior integration into an EAM tool. Semantic limitations regarding LM2F must also be considered in future work, and Schweda’s building block approach (Schweda, 2011) could provide a useful starting point. A possible extension would be to relate other meta-model and model evolution approaches towards the architecting of the digital transformation. Furthermore, gathering more empirical evidence from further applications of LM2F to other industries could add more representativeness to the results, hence reinforce the value of the approach in practice.

Acknowledgements

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


