Using Weaving Models in Metamodel and Model Co-Evolution Approach

F. Anguel  
Department of informatique, El Taref University, El Taref, Algeria.  
LISCO Laboratory. Badji Mokhtar University, Annaba, Algeria  
fanguel@yahoo.fr

A. Amirat  
Department of informatique. Mohammed Chérif Messaadia University, Souk-Ahras, Algeria  
abdelkrim.amirat@yahoo.com

N. Bounour  
LISCO Laboratory. Badji Mokhtar University, Annaba, Algeria  
nora_bounour@yahoo.fr

Abstract—In Model Driven Engineering, analogously to any software artifact, metamodels are equally prone to evolution. When a metamodel undergoes modifications, all the related artifacts must be accordingly adapted in order to remain valid. Manual co-evolution of models after these metamodel changes is error-prone. In this setting, this paper introduces a semi-automatic process for the co-evolution of models after metamodel evolution. The process is divided in four main stages: at the differencing stage, the changes to the metamodel are detected. After that these changes are linked with the original model elements and represented in a weaving model which serves to generate a transformation used in the last stage in order to obtain the evolved model. Contributions of this paper include the automatic co-evolution of breaking and resolvable changes and the assistance to the model developer in the co-evolution of breaking and un-resolvable changes.

Keywords— Model Driven Engineering; metamodel evolution; metamodel differences; weaving model; model co-evolution; migration.

I. INTRODUCTION

Models have been used in various engineering fields to help managing complexity and represent information in different abstraction levels, according to specific notations and stakeholder’s viewpoints. Model-Driven Engineering (MDE) is increasingly emerging as a discipline which strictly prescribes designers to develop software in terms of models rather than programs. According to this perspective, models are considered first-class artifacts throughout the software life-cycle. In the field of MDE, a model is an artifact that conforms to a metamodel and represents a given aspect of a system [1]. Metamodels are models that describe the structure of models [1]. Models are managed by a wide-range of tools including model transformations. Thus, metamodels are cornerstones upon which many artifacts and functionalities are defined.

Evolution is unavoidable and affects the whole software lifecycle [2]. Analogously to software, metamodels are subject to evolutionary pressure too which range from technological and business changes [3]. However, changing a metamodel might invalidate related artifacts that are defined in terms of it i.e. models and transformations, whose validity must be restored. Admittedly, when manually operated the adaptation is error prone. Recently, several approaches addressing the problem of co-evolution have been proposed. To the best of our knowledge, there is no Model Driven Engineering (MDE) approach that provides enough generic mechanisms to automate the co-evolution of models.

In this proposal, in order to automate the co-evolution of models it is necessary to discover the different kinds of relationships (links) between metamodel changes and model elements. These links must be saved in another model called weaving model. This model can be validated or modified by human expert. The weaving model conforms to the extensions of a weaving metamodel. It contains links that are used to produce model transformation to automatically migrate models. The overall process is the following:

- Two input versions of a metamodel are provided : the original version MM1 and the evolved one MM2;
- Calculate metamodel differences, this step results the difference model (MD).
- A sequence of matching operations is executed. They produce a weaving model with a set of links between MD and M1 the initial model that conforms to MM1;
- The human expert verifies the links and validate them;
- A transformation model is generated based on the set of links;
- The transformation produced is used to evolve the model conforming to MM1 into a model conforming to MM2.

This paper is organized as follows. Section 2 presents the core MDE concepts used in this paper. Section 3 presents weaving metamodel extensions that capture different kinds of relationships between models. The general process of the production of model transformations used to adapt a specific model to an evolved version is presented in section 4. Section5 present an application example with Petri net metamodel related works are discussed in section 6. Section 7 concludes.
II. BASIC CONCEPTS

In this section we present the central MDE definitions used in this paper. The basic assumption in MDE is to consider models as first-class entities. An MDE system basically consists of metamodels, models, and transformations. A model represents a view of a system and is defined in the language of its metamodel. In other words, a model contains elements conforming to concepts and relationships expressed in its metamodel. A metamodel can be given to define correct models. In the same way a model is described by a metamodel, a metamodel in turn has to be specified in a rigorous manner; this is done by means of meta-metamodels. This may be seen as a minimal definition in support of the basic MDE principle “Everything is a model” [1]. The two core relations associated to this principle are called representation “Represented by” and conformance “conform To”. In this respect, OMG [4] has introduced the four level architecture which organizes artifacts in a hierarchy of model layers (M0, M1, M2, and M3). Models at every level conform to a model belonging to the upper level. M0 is not part of the modeling world, so the four level architecture should more precisely be named (3+1) architecture [1] as depicted in Fig. 1. One of the best-known metamodels in the MDA is the UML metamodel; MOF (Meta-Object Facility) is the metametamodel of OMG that defines UML [4].

A. Metamodel Evolution

Metamodels can be considered one of the constituting concepts of MDE, since they constitute the languages by which a given reality can be described in some abstract sense [1]. Metamodels may evolve in different ways, due to several reasons. Furthermore, parts of the metamodel are redesigned due to a better understanding or to facilitate reuse. From the work presented in [5], we take the following example which illustrates the evolution of a simplified Petri net metamodel as depicted in Fig. 2. A Petri net consists of any number of Places and Transitions. Each transition has at least one input and one output place (src and dst association roles, respectively). The initial metamodel (MM1) captures these facts. In a next step (MM2) is evolved to (MM2), in which Net is restricted to comprise at least one Place and one Transition. Moreover arcs between places and transitions are made explicit by extracting Arc-src and Arc-dst metaclasses. This refinement permits to add further properties to relationships between places and transitions.

B. Model co-evolution

Due to changing requirements and technological progress and like other software artifacts, metamodels evolve over time during their life cycle [3]. The addition of new features and/or the resolution of bugs may change metamodels, thus causing possible problems of inconsistency to existing models which conform to the old version of the metamodel and may become not conform to the new version. Therefore to maintain consistency, metamodel evolution requires model adaptation, i.e., model migration, as shown in Fig. 3.

These two steps are referred as model and metamodel co-evolution. Furthermore, model adaptations should be done by means of model transformations. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to a given metamodel.

Changes occurring on a meta-model may have different effects on the corresponding models. In this respect metamodel changes are classified as follows [6] :

- Not breaking changes, changes occurring in the metamodel don’t break the models conformance to the metamodel.
- Breaking and resolvable changes, changes occurring in the metamodel do break the models, which can be automatically resolved.
- Breaking and unresolvable changes, changes do break the models and cannot be automatically resolved and user intervention is required.

Evolutions of Petri net metamodel illustrated so far, can invalidate existing instances (i.e. models); therefore, each version needs to be analyzed to comprehend the various kind of updates it has been subject to and, eventually, to adapt corresponding models.

C. Weaving Model

Weaving model is other important operation in MDE. Its primary objective is to handle fine-grained relationships between elements of distinct models, establishing links between them. These links are captured by a weaving model [7]. In [7], a generic way is provided to establish model element correspondences by proposing solution where weaving operations are introduced; these are considered as models that conform to a weaving metamodel.
A weaving model can be seen as a means for setting fine-grained relationships between models and executing operations on them [1], weaving models enable the creation of abstract links between model elements. The core weaving metamodel is depicted in Fig. 4.

The core metamodel has elements with information about link types, link endpoints and element identifications.

- **WElement** is the base element from which all other elements inherit.
- **WModel** represents the root element that contains all model elements. It is composed by the weaving elements and the references to woven models.
- **WLink** expresses a link between model elements, i.e., it has a simple linking semantics.
- **WLinkEnd** defines the link endpoint types. Every link endpoint represents a linked model element. It allows creating N-ary links.
- **WElementRef** elements are associated with a referencing function. This function takes as parameter the value of the ref attribute and it returns the linked element.

Defined link classes contain two references to save the source and target elements named respectively “source” and “target”, except **WAddedLink** which do not require to be related to the corresponding entities in the input model since they are not expected to exist. The extended part of the core weaving metamodel to support metamodel differences is depicted in Fig. 5; we called the resulted metamodel **Generic Weaving Metamodel for Differences representation** “GWMM4D”. The creation of the weaving metamodel is a manual task, based on design decisions. Consequently, its specification is a key process of our approach.

### III. SPECIFICATION OF GENERIC WEAVING METAMODEL FOR DIFFERENCES REPRESENTATION

In fact this task is general and the produced result is applicable to every metamodel. The weaving metamodel is created as extension of a core weaving metamodel presented in Fig. 4. Therefore a weaving model (WModel) consists of elements (WElement) related through difference links (WDLINK), according to the different kind of differences involved in metamodel evolution. The class Element is a concrete extension of (WLinkEnd). (InitialElement) and (DiffElement) specialize the Element concept. They enable referring respectively initial model elements and difference model elements. Moreover, (WDLINK) is specialized into (WAddedLink), (WDeleteLink), (WChangeLink) and (WMoveLink), to support the four kind of changes or differences detected with EMFCompare nameded respectively; addition, deletion, change and Move. To represent elements to be conserved, we use UnchangedLink.

Defined link classes contain two references to save the source and target elements named respectively “source” and “target”, except WAddedLink which do not require to be related to the corresponding entities in the input model since they are not expected to exist. The extended part of the core weaving metamodel to support metamodel differences is depicted in Fig. 5; we called the resulted metamodel **Generic Weaving Metamodel for Differences representation** “GWMM4D”. The creation of the weaving metamodel is a manual task, based on design decisions. Consequently, its specification is a key process of our approach.
same metamodel, and then, based on those differences, semi-
automatically generate model migration. As depicted in Fig. 6,
the process of adapting models is organized in four steps:

1- Calculate metamodel differences;
2- Generate weaving model differences;
3- Generate model migration transformation;
4- Calculate resulting model.

In the following sections, we explain in detail these
different phases of the proposed process.

A. Calculate metamodel differences

This stage takes as input both the original metamodel
(MM1) and the evolved meta-model (MM2), and infers the set
of changes. To this end, we use EMF Compare tool [9]. We
have used EMF because it is an integral part of the Eclipse
framework [10] that can be used for comparison of Ecore
based models and its ability and accuracy has been proven in
several works. This tool takes two models as input and obtains
the differences along the Difference metamodel. In our case,
given two versions MM1 and MM2 of the same metamodel,
we calculate the differences between the evolved and original
metamodel. The result of this step is presented as a difference
model (DM). Hence, a difference model formalizes all kinds of
modifications, i.e. non-breaking, breaking resolvable and
un-resolvable ones. Difference model (DM) conforms to
EMFcompare differences metamodel. We note that difference
model enables the opportunity to apply differences to any
model conforming to the same metamodel the original ones
conforms to. The process of this phase is called differencing
process as shown in Fig. 6.

B. Generate weaving model differences

This step consists to generate a weaving model for
differences related to the initial model that conforms to the
original metamodel. This step follows a matching algorithm.
The goal of this phase is to discover the impact of metamodel
evolution on the initial model. This means that the initial
model M1 is matched against the difference model (DM) to
build correspondences between entities, which is to recognize
the portion of the initial model intersecting the difference one.
Those results are recorded in a weaving model (DWM) that
conforms to (GWMM4D). The resulting weaving model
should be analyzed by a designer, and can be refined in order
to improve its accuracy, and it is exploited in the following
step.

Matching algorithm traverses the graph representation of
M1 (tree representation, where the edges are instances of the
containment relation, is traversed), and for each model element
checks whether the metamodel element that that element
conforms to has changed (i.e. attached as left target element
with Diffelement in the difference model), if this was the case,
based on the kind of the difference contained in Diffelement
(attribute Kind) in the difference model DM, the discovered
mapping is stored in (DWM) as follows:

- Creating in DWM an instance of initial element
  containing features of the tested model element in M1
  and an instance of Diffelement corresponding the
  matched element in the difference model ("left target").
- Creating an instance of WDLink type according to the
  kind of the detected difference specified in Diffelement
  the created object is an instance of WDeletelink,
  WChangelink or WMovelink.
- Creating two references the first one labeled source
  between the created instance of Diffelement and the
  created instance of WDLink and the second one labeled
  target between the created instance of M1 model
  element and the created instance of WDLink.

The weaving model obtained by the automated weaving
mechanism can be refined (in order to improve its accuracy)
through manual intervention which could be required to establish custom mappings. We referenced this process by matching process in Fig. 6.

**C. Generate model migration transformation**

The generation of model migration transformation is the last phase in the overall process. We propose higher-order transformations (HOT) that interpret the different kinds of links captured by the difference weaving model (DWM), This HOT take as input the difference weaving model, and produce a model transformation (Tmig) as shown in Fig. 6. The transformation is written in ATL [11]. We choose ATL because it provides a simple syntax adapted for model transformations. The *model transformation interprets manipulations contained in the weaving model to perform the required changes.

Therefore, Tmig transformation can be applied to the initial model M1 in order to obtain the evolved model M2 with respect to the correspondences stored in the difference weaving model.

Tmig transformation implements the rules to apply on a model M1 additions, deletions and changes specified in the model (DM) with respect to the links specified in DWM conforming to GWMM4D, i.e. the weaving meta-model proposed to specify differences between metamodels as depicted in Fig. 6. Principal Tmig transformation rules are summarized below:

- **Unchanged rule**: the rule copies the instances of initial model element which have to be the same both in M1 and M2. Thus, these elements are represented by all elements attached with

- **Added rule**: for each add link in DWM, the rule creates in M2 an instance of initial model element if it is not yet created, then setting the structural features according to DiffElement attached as source reference with the same link.

- **Changed rule**: for each update or move link in DWM, the rule creates in M2 an instance of initial model element if it is not yet created, then updates the structural features according to modifications specified in DiffElement attached as source reference with the same link.

- **Delete rule**: for each delete link in DWM, the rule checks if initial model element is not created in M2. In positive cases nothing happens (hence, not copied in the evolved target M2), otherwise this element is removed from M2.

The specified rules implemented with ATL represents the HOT transformation (Fig. 7.) which when executed with specific initial model provide the Tmig transformation that calculate automatically the evolved model conforming to the new version of the metamodel. We called this double steps transformation and migration process.

**V. APPLICATION EXAMPLE**

This section presents our metamodel evolution approach illustrated by means of a Petri net example. We Consider an EMF Petri net metamodel (MM1) as shown in Fig. 2. An evolved metamodel with appropriately deduced node and edge names is shown as (MM2) in Fig. 2.

The following ATL code represented in Fig. 8. is a fragment of Tmig transformation.

**W. RELATED WORKS**

Over the last few years, the problem of metamodel evolution and model co-evolution has been investigated by several works like [2], [6], [8-15]. Currently, there are several approaches that focus on resolving inconsistencies occurring in
models after metamodel evolution, Rose et al. [16] propose a classification of model migration approaches. This classification highlights three ways to identify needed model updates: manually, based on operators, and by using metamodel matching. When manually approaches, like in [12-14], updates are defined by hand. In operators based approaches, like [5],[15], metamodels changes are defined in terms of co-evolutionary operators. Those operators define conjointly the evolution on the metamodel and its repercussions on the models. Finally, in metamodel matching, like [2], [6], [8], versions of metamodels are compared and differences between them are used to semi-automatically infer a transformation that expresses models updates. Manual specification approach like Flock [14] is very expressive, concise, and correctness is also assured [14] but finds difficulties with large metamodels since there is no tool support for analyzing the changes between original and evolved metamodels. Operator based approaches like [18] ensure expressiveness, automatically, and reuse, it was been perceived as strong in correctness, conciseness and understandability [19] but its lack is in determining which sequence of operations will produce a correct migration. In turn matching approaches like [2] uses heuristics which weaken its correctness. Analysis of existing model co-evolution approaches has yielded guidance for defining some requirements to a novel approach. We focus to minimize as far as possible the user effort to migrate models by automatically generating difference model weaving that save relationships between metamodel changes and a specific model to be migrated and based on this result a model transformation to migrate input model. Automaticity is also increased by automatically copying of unchanged elements as in flock [14]. Human intervention to validate weaving models permits to increase expressivity and the correctness of the proposed solution. We consider also that our solution has a generic dimension through the generic weaving metamodel for differences representation and also through the HOT that generates. We focus also in our approach to ensure correctness, conciseness and understandability.

VI. CONCLUSION

In this paper we have highlighted the crucial role that weaving models can play in the process of metamodel and model co evolution. We have concentrated on the proposition of a generic weaving metamodel for differences representation. The solution proposed here focuses on the automaticity dimension as in our previous work [20] and aim also to obtain a generic approach. So contributions of this paper are the following: We have proposed a methodology to semi-automate the co-evolution of models. The methodology is based on using model weaving to store relationships between a model conforming to an original metamodel and the evolutions or changes that this metamodel undergoes. We propose a metamodel-based method that exploits the information from the set of input metamodels and from the weaving metamodel to generate a model transformation to migrate the initial model to the new one that conforms to the evolved version of the metamodel. Creating weaving models is executed automatically but human intervention is authorized to adjust the results. compared to existing approaches we note that our proposal increase automaticity, expressiveness and correctness and understandability are also assured.

In perspective, to validate our approach, we will firstly develop a prototype using theoretical example presented in the literature. Then, we will realize in the future an experiment using large metamodels, to demonstrate the feasibility and scalability of our approach in real world scenarios.

REFERENCES