A MOF-BASED CHANGE META-MODEL

Dhikra Kchaou  
Mir@cl Laboratory, Sfax University Sfax, Tunisia  
Dhikra.kchaou@fsegs.rnu.tn

Nadia Bouassida  
Mir@cl Laboratory, Sfax University Sfax, Tunisia  
Nadia.Bouassida@isimsf.rnu.tn

Hanene Ben-Abdallah  
Mir@cl Laboratory, Sfax University Sfax, Tunisia  
Hanene.BenAbdallah@fsegs.rnu.tn

Abstract: Software evolution induces various types of changes on the models developed during the various steps of the software lifecycle. The induced changes often require the re-conduct of several development activities to ensure the consistency and correctness of the updated/new models. To assist developers in limiting the development activities to be reapplied, and hence to reduce the resulting costs, it is important to manage the change impacts on the development activities. Towards this aim, we propose a change meta-model that is independent of any software modeling language and covers several types of model changes. To ensure the first characteristic of our change meta-model, we define it based on the Meta Object Facility (MOF) standard meta-meta-model; this latter can be used to define various modeling languages amongst which UML. In addition, we ensured that it covers all types of changes proposed in existing change meta-models. Furthermore, we illustrate how our change meta-model can be used to trace the impact of changes on model transformation, an important activity in model-driven software development.

Keywords: change management, change model, change impact, MOF, model transformation.

1. INTRODUCTION

Software systems, and in particular long-lived ones, are inevitably subject to continuous evolution and changes. Changes may be the results of the emergence of new requirements, the improvement of existing features, the reparation of errors, etc. They affect different types of models during the various steps of the software development cycle, e.g., data models, behavioral models, domain models… On the other hand, the complexity of today’s software systems makes the adaptation of their models to change a difficult, tedious and expensive task. Hence, changes should be managed and planned efficiently to produce consistent models with reasonable costs/efforts.

One step towards attaining this goal is to define, precisely, the various types of changes in terms of their impact on the modeling elements. Such a precise definition provides for delimiting the software development steps affected by each change. For instance, in the case of model-driven software engineering (cf., [22]), a precise change definition can be used to delimit the transformations that must be redefined and/or re-executed in order to update the old models. Consequently, the developers would focus on only the model parts and transformations to be updated/re-executed and preserve those parts of the models that are unaffected by the change.

In this context, the change definition can be specified through a change meta-model that describes both all different types of changes a model may undergo and the software modeling elements they affect. This latter information can be used to trace the effects of each change on the model elements and the development activities that should be redone.

Given these advantages of change meta-models, several researchers proposed different change meta-models that can be classified into two types: modeling language dependant meta-models (cf., [4]) and modeling language independent meta-models (cf., [3], [8], [11], [14], [15], [25]). One inconvenience of the first type of meta-models is that their instance change models can be applied only on models expressed in the base modeling language; that is, they cannot be reused as a "patch" for other types of models. To overcome this limit, the second type of change meta-models uses the generic concepts of model element and change type.

In addition, some of the proposed meta-models specify: how the change operations can be applied, e.g., in elementary (cf. [8]) or composite steps (cf. [13], [15]); whether their effect can be breaking with old models (cf. [7]); whether the potential conflict they may introduce is detectable …

Overall, the proposed meta-models agree upon a set of change types. However, as presented in this paper, a few change types must be redefined in order to ensure a more precise traceability, a necessary thing in change impact management. For instance, a composite change must specify the order of the change operations; in fact, the order may influence the way model transformations are re-executed. In addition, tying the meta-model to the Meta Object Facility (MOF) meta-meta model provides for a more precise definition of the change operations. On the other hand, being MOF-based, our change meta-model remains applicable to all MOF-based modeling languages.

Besides validating a given change, our challenge is using the change meta-model in managing its impact on the various developed software models and development activities. More specifically, we would like to have a change meta-model that provides for the definition of the activities that need to be reapplied. For instance, in the case of model-driven engineering, our change meta-model should delimit the transformations that must be
partially/completely reapplied and/or replaced in order to account for an incoming change. Thus, when a change occurs, the developer would have to update only the affected transformations and avoid the regeneration of all transformations, which is an expensive task. Several works (e.g., [11], [13], [17], [27], [28], [29]) studied change impact on model transformations. However, they do not address the problem of limiting the re-execution of all transformations, efficiently and with minimal cost. In this paper, we illustrate how our change meta-model offers sufficient traceability to delimit the affected transformations, i.e., those that must be re-defined and/or cancelled. The remainder of this paper is organized as follows. Section 2 first overviews existing approaches to model changes; secondly, it presents the set of necessary criteria for change definition. Section 3 presents our change meta-model and illustrates it through an example. Section 4 shows how this meta-model can assist in the management of change impact on model transformations. Finally, section 5 concludes with a summary and outlines future works.

2. RELATED WORK

Several approaches have been interested in managing software evolution through the identification of changes but from different perspectives. Initial works (e.g., [1], [18], [17]) did not use models when presenting differences, thus being at a low level of abstraction. As a result, many model-based approaches have appeared.

2.1 NON MODEL BASED APPROACHES

Alalen and Porres [1], Ohst et al. [18] and Xing et al. [27] present difference types among UML models. In their work, they focused on detecting the occurrences of a change by using matching algorithms. Nevertheless, these approaches detect changes at a low level of abstraction and do not allow the explicit representation and visualization of changes.

Other works represent and visualize model differences based on edit scripts [1][17] and coloring techniques [18]. However, they have drawbacks since they lack abstraction and compositionality, which prevent them from being represented by suitable meta-models and processed with standard modeling platforms [8]. To avoid these drawbacks, model-based approaches have emerged aiming to represent changes at a high level of abstraction.

2.2 MODEL BASED APPROACHES

Table 1 presents different model-based approaches. As summarized in this table, the proposed change meta-models differ in terms of their dependence on the meta-model elements which reflects their design view/perspective. In all these cases, a change meta-model is defined to present changes.

Some works are dependent to the UML modeling language. For instance, [4] treats the change between UML models by adding a class named “change” related to the Model Element.

On the other hand, EMF compare [25] offers a tool to compute differences between EMF models. It is based on an EMF change model describing elements (Element, Attribute, Reference) that may be changed as well as change types affecting an element (Addition, Deletion, Update, Movement). Moreover, it introduces the concept of Conflict and Group of change to deal with, respectively, divergences in the different models and changes which may depend on each other. Despite the importance of this work, it does not present the composite change aspect.

Rath et al. [21] introduce the concept of change history model (CHM) to preserve the traceability. It offers a generic change history meta-model using the VPM [26] meta-modeling approach, which uses two basic elements: entities and relations. Change operations that affect these elements are: creations, deletions, updates with setvalue and setname operation to indicate the old and the new value or name of the changed element, and move with SetRelationTarget and SetRelationSource to represent retargeting and resourcing operations.

Other works presented change meta-models that affect a model independently of the changed and source modeling languages. For instance, [8] presents a meta-model independent approach to represent the differences between models. It introduces three classes to represent additions, deletions, and changes related to each element of any meta-model. Despite its model-independence, this approach does not provide for changes traceability and changes list. For a composite change, it indicates that a difference model must be compositional but it does not present a solution for composite differences.

Könemann [14][15] also presents a meta-model independent approach for the representation of model differences, its meta-model is agnostic of the base meta-model i.e. the meta-model the base models conform to. According to this work, models may consist of arbitrary elements, having attributes, references, and maybe other properties. It presents their meta-model for model-independent difference in three class-diagrams. The first one contains the root element for differences, IndependentChangeModel, as well as several classes for the actual changes (elements, attributes, or references can be added, removed, or changed). The proposed notation presents important concepts relative to change like traceability and change group. However, it does not present the composite change concept.

Burger and Gruschko [7] introduce a change meta-model describing the transformation of one version of a MOF-based meta-model (MOF 1.4) to another. Instances of this meta-model describe an actual change to a meta-model as a sequence of single change operations, contained in a ChangeSequence element. All changes in this meta-model can be expressed as a sequence of either additions or deletions of elements and links, or modifications of a property.

Finally, [3] present RCVDiff which is a stand-alone tool for representation, calculation and visualization of model differences. The difference meta-model shows that an element can be changed, deleted or added. Element change contains the differences between attributes, references and sub-objects of objects in model A and their corresponding objects in model B. Note here that the change of an element is not considered as well as its movement.

Overall, most of existing works are initial investigations in the area. Moreover, they tackled the change from different views. Thus, none of the proposed notations expresses all of the indispensable concepts for managing the changes. This motivated our work in proposing a change meta-model that also satisfies the criteria stated in [8].
(model based, model independent, compositional, transformative, and minimalistic).

3. CHANGE META-MODEL

Our main objective in defining a change meta-model is, on the one hand, to cover all necessary concepts for change modelling and, on the other hand, to use it in analyzing change impact on model transformations. As mentioned in the introduction, we took care in defining all possible changes affecting the elements independently of a particular modeling language. For this, we relied on the MOF meta-model 2.0 [19] since it is the base of several meta-models like UML and Petri nets. As shown in Figure 1, the Elementchange class represents the change operations. Furthermore, each change operation is related to a modeling element to provide for change impact management. In fact, the change affects either an element or its characteristics known in MOF as Feature. A modeling element can be a class in a UML model, a state in a Petri nets model, etc. An element characteristic can be an attribute in a UML model, the number of tokens in a place in a Petri nets model, etc.

An element may be added, deleted, updated or moved. Moreover, for traceability purposes, our meta-model indicates the old and the new value if an element is updated and the old and the new parent if the element is moved.

On the other hand, a Feature may be added, deleted, changed or moved. Our meta-model keeps track of the old value and the new value if the feature is changed and the old element and the new element, containing this feature, if the feature is moved from an element to another. Note that we decided to use the change operation move while it can be represented by a deletion followed by an addition to facilitate the change modeling and to ensure the traceability. Also note that, in our meta-model, we distinguish between the changes of a structural feature (Property) and a behavioral feature (Operation).

Note that our meta-model can be easily extended to present the change of other modeling concepts. In our change meta-model (Figure 1), a relationship between elements may be added, deleted or changed. As a result, we indicate the endpoints of a new relation, of a deleted relation and the old and the new endpoint of the changed source or target relation. We indicate also the type of the added relationship and the end type of the updated relationship that can be an association, an owning association (composition) or a navigable association [19] in the case of UML models.

The Compositechange class provides for the specification of the composition of changes. A change may be an atomic change composed of a unique element “AtomicElementChange”, a change of an element feature “FeatureChange”, a change of a relation “RelationshipChange” or a composite change. The Compositechange class models the change composition. The composition tie indicates that a composite change can be composed of one or several element changes, feature change or relationship change. Moreover, we model the ordered list of changes in the class ListChange in order to allow any reproduction of the changed model from the original model and the change list; As a consequence, it would be possible to reproduce the original model from the changed one and the change list. The change list is used also for documentation and visualization of changes [13].

Furthermore, as illustrated in Figure 1, we take into account the kind of change which can be none-breaking, breaking and resolvable and breaking and not resolvable [7]. The none-breaking changes do not affect existing models. Breaking and resolvable changes affect models and they can be resolved by an algorithm. Finally, Breaking and not resolvable changes affect models and require a manual interaction to solve them.

Let us illustrate how our change meta-model can be used to specify model changes. Consider the example presented in Figure 2 where the source model (Figure 2.a) indicates that a HTML document is composed of a HTML Form, a HTML List and a HTML Combo. On the other hand, a HTML form is composed of a HTML List and a HTML Combo. This source model can be improved to obtain the model shown in Figure 2.b through the following, ordered list of changes:

1. Addition of the HTMLDocElem
2. Addition of the attribute name to the HTMLDocElem
3. Addition of the operation add()
Figure 1. The proposed change meta-model
Operation in localizing the transformation.

Our change management approach consists in, firstly, logging facility within the editor of the modeling tool.

In addition, suppose that a change \( M \) was applied to \( M_{source} \) to produce \( M_{target} \). (\( M_{source} \) is an instance of our change meta-model. It can be created through a logging facility within the editor of the modeling tool.).

Our MOF based model conforms to a meta-model defined based on MOF. In addition, suppose that a change \( M_c \) was applied to \( M_{source} \) to produce \( M_{target} \). (\( M_{source} \) is an instance of our change meta-model. It can be created through a logging facility within the editor of the modeling tool.).

That is, we illustrate how we can use a change model to identify which transformations must be updated/re-written and avoid rewriting and/or re-executing all transformations from scratch. Let us assume that we have a set of transformations between two models \( M_{source} \) and \( M_{target} \) conforming to a meta-model defined based on MOF. In addition, suppose that a change \( M_c \) was applied to \( M_{source} \) to produce \( M'_{source} \). (\( M_{source} \) is an instance of our change meta-model. It can be created through a logging facility within the editor of the modeling tool.).

Our change management approach consists in, firstly, localizing the transformations affected by each change operation in \( M_c \), secondly, rewriting and/or re-executing these transformations. To illustrate change impact on transformations, let us consider the transformation of the source model \( M \) presented in Figure 2.a to XML (the target model \( M' \)) using the transformation rules \( T(M) \). Informally, these transformation rules represent the following operations inspired from [2]:

- **T1(M):** The class instance “HTMLDoc” is transformed to an XMLElement. The XML element has two XMLAttribute. The first is the name of the class which has a value of “HTMLDoc”. The second is a unique identifier id= “UMLClass.HTMLDoc”.

- **T2(M):** The operation “dump()” is transformed to an XMLElement. The XML element has two XMLAttribute.

Table 1. Overview of change model related work

<table>
<thead>
<tr>
<th>Modeling work</th>
<th>Change Concepts</th>
<th>Affected Elements</th>
<th>Change operations</th>
<th>Atomic/ Composite change</th>
<th>change Group</th>
<th>Change effects</th>
<th>Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cichetti et al. [8]</td>
<td>Meta Class of a given meta-model</td>
<td>Add, Delete, Change</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brand et al. [3]</td>
<td>Element Attribute Reference</td>
<td>Add, Delete, Change, Move Change(old, newvalue) Change(old, newreference)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Konemann [14] [15]</td>
<td>Element Attribute Reference</td>
<td>Add, Remove, Update(old, newValue) Add, Remove, Update(old, newReference)</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burger et al. [7]</td>
<td>Element Link, Property</td>
<td>Add, Delete Change</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rath et al. [21]</td>
<td>Entity relation</td>
<td>Create, Delete, Update(setvalue, setname), Move(newparent) Create, Delete, Move(setrelationtarget, setrelationsource)</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMF Compare [25]</td>
<td>Element Attribute Reference</td>
<td>Add, Delete, Change, Move, Conflict</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Briand et al. [4] [5]</td>
<td>UML Model Element</td>
<td>Change</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our MOF based change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meta-model</td>
<td>Element Feature Relationship</td>
<td>Add, Delete, Update(old, newvalue), Move(old, newparent) Add, Delete, Update(old, newvalue), Move(old, newelement) Add, Delete, Update(source, target end)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Change model instance for the example of Figure 2
XMLAttributes: The name of the operation which has a value of “dump()”, and a unique identifier id=’"HTMLDoc.dump()"’.

- **T3(M)**: The association “comp1” is transformed to an XMLElement. An association is comprised of two association ends which also need to be related to XML:
  - “end1” is related to an XMLElement named “associationend” having two XMLAttribute to represent the name of the endpoint 1 “HTMLForm” and the unique identifier of the association end id=’"UMLClass.HTMLForm.end1"’.
  - “end2” is related to an XMLElement named “associationend” having two XMLAttribute to represent the name of the endpoint 2 “HTMLDoc” and the unique identifier of the association end id=’"UMLClass.HTMLDoc.end2"’.

- **T4(M)**: The class instance “HTMLForm” is transformed to an XMLElement. The XML element has two XMLAttribute. The first is the name of the class which has a value of “HTMLForm”. The second is a unique identifier id=’"UMLClass.HTMLForm"’.

- **T5(M)**: The attribute “Name” is transformed to an XMLElement. The XML element has two XMLAttribute. The first is the name of the class which has a value of “Name”. The second is a unique identifier id=’"HTMLForm.Name"’.

Now, let us update the transformation rules T(M) to account for the change example presented earlier (see Figure 3). Note that, the change model informs us about the element undergoing change and allows us to localize this change.

Thus, we can determine the transformation Ti(M) related to the changed element. For instance, the change model indicates no change operation on the HTMLDoc element; thus, transformation T1 is unconcerned with this change. However, the change model indicates that the method dump() is deleted from the HTMLDoc element; thus the transformation T2 must be deleted as well as all corresponding XML elements created by T2 in the Target model. The change model instance makes a traceability link between the element (dump()) and transformation (T2), since it presents the change type and the elements affected by this change. In addition, the composition “comp1” is deleted, and thus T3 must be updated since it has been identified as the transformation of “comp1” to XML. The transformations T4 and T5 will be preserved.

5. Conclusion and future work

This paper first proposed a MOF-based change meta-model which includes necessary concepts for the presentation of evolution in a software system. It then presented an example showing that our change model covers all change types and presents them easily. The example showed that the change model facilitates the visualization of changes especially the composite type. In addition, it is used to illustrate how it can be used to delimit the impact on model transformations.

In our future work, we are interested in examining the change impact on model transformations and in assisting the developer in re-writing the impacted transformations.

References


