Supporting Users to Manage Breaking and Unresolvable Changes in Coupled Evolution

Juri Di Rocco Davide Di Ruscio
Alfonso Pierantonio
Department of Information Engineering Computer Science and Mathematics
University of L'Aquila
name.lastname@univaq.it

Ludovico Iovino
Gran Sasso Science Institute, L'Aquila, Italy
ludovico.iovino@gssi.infn.it

Abstract

In Model-Driven Engineering (MDE) metamodels play a key role since they underpin the specification of different kinds of modeling artifacts, and the development of a wide range of model management tools. Consequently, when a metamodel is changed modelers and developers have to deal with the induced coupled evolutions i.e., adapting all those artifacts that might have been affected by the operated metamodel changes. Over the last years, several approaches have been proposed to deal with the coupled evolution problem, even though the treatment of changes is still a time consuming and error-prone activity. In this paper we propose an approach supporting users during the adaptation steps that cannot be fully automated. The approach has been implemented by extending the EMFMigrate language and by exploiting the *user input* facility of the Epsilon Object Language. The approach has been applied to cope with the coupled evolution of metamodels and model-to-text transformations[1].

1. Introduction

Metamodels play a key role in any *metamodeling ecosystem* [4] since they underpin the development of a wide range of modeling artifacts and tools including models, model transformations, textual and graphical editors, and code generators. Similarly to any software component metamodels are expected to evolve during their life-cycle [5] and as a such it is necessary to deal with the *coupled evolution problem* i.e., managing the ripple effects that metamodel evolutions might have on all the existing related artifacts. Depending on the corrupting or not-corrupting effects, metamodel changes can be classified as *i) non-breaking*, if they do not break the relations between the evolving metamodels and existing artifacts defined on them, *ii) breaking and resolvable* if they break such relations even though the affected artifacts can be automatically co-adapted, and *iii) breaking and unresolvable* if they break the existing relations which cannot be automatically recovered because of lack of information [1]. In the last case, user intervention is required [5], e.g., to set the value of new obligatory attributes or references.

Over the last years, the coupled evolution problem has been largely investigated in MDE and different approaches have been proposed. Some authors have investigated the metamodel/model coupled evolution problem, and propose techniques and tools able to adapt models that are no longer conforming to the initial version of the changed metamodel and thus that have to be adapted with respect to its new version [1, 9, 10, 16, 17, 20]. Similar approaches have been proposed to deal with other coupled evolution problems, including model transformations [2, 6, 15], graphical and textual concrete syntax editors [3, 7], and OCL queries [13]. All these approaches aim at automating the management of non-breaking, and breaking and resolvable changes, whereas the management of breaking and unresolvable changes is still a challenging and error-prone activity.

In this paper we propose an approach enabling users to give input during the adaptation phases when needed. This permits to overcome the limitations of existing coupled evolution techniques that currently provide modelers with limited support for solving the lack of information induced by breaking and unresolvable changes. In fact, these are usually managed by applying default migration actions according to predefined heuristics [5]. In some cases partially migrated artifacts are produced and they have to be finalized by modelers to fix those parts invalidated by the unresolvable metamodel changes. The proposed approach is supported by an extension of the EMFMigrate language [5] that is implemented atop of the Epsilon platform[2]. Even though the approach and the supporting tools are general and can be applied to adapt any kind of modeling artifacts [4], in this paper we consider an explanatory scenario requiring the adaptation of Acceleo-based model transformations.

[1] This research was supported by the EU through the Model-Based Social Learning for Public Administrations (Learn Pad) FP7 project (619583)

[2] http://www.eclipse.org/epsilon/

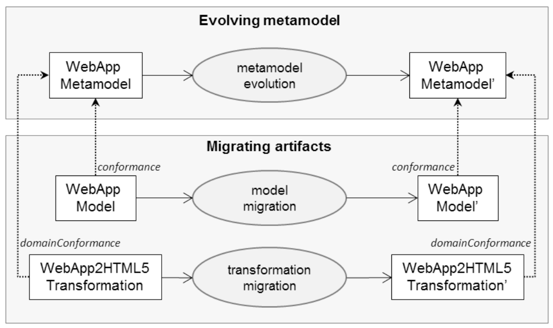


Figure 1. A coupled evolution scenario

The paper is organized as follows. In Section 2 the coupled evolution problem is presented and we discuss the impact that breaking and unresolvable metamodel changes have on Acceleo-based transformations. In Section 3 we propose an adaptation process that makes explicit the activities involving users during the migration of affected artifacts. The supporting tool consisting of an extension of EMFMigrate, and its application on a concrete scenario are also presented. Related work is described in Section 4, and conclusions and research perspectives are given in Section 5.

2. Motivating scenario

The problem of restoring the consistency among evolving metamodels and existing artifacts is intrinsically difficult. Figure 1 shows a simple coupled evolution scenario consisting of the `WebApp` metamodel and two different artifacts depending on it, i.e., the `WebApp Model` and the `WebApp2HTML5` model-to-text transformation. The `WebApp` metamodel consists of modeling constructs specifically conceived to develop Web applications which can be generated by means of the provided `WebApp2HTML5` transformation out of source `WebApp` models. As shown in the upper side of Fig. 1 the `WebApp` metamodel might require changes in order to address unforeseen requirements or to better represent the considered application domain. Such changes might invalidate existing artifacts, as for instance the `WebApp Model` and the `WebApp2HTML5` transformation on the left-hand side of Fig. 1 that might have to be adapted to recover their relations (i.e., *conformance* and *domain conformance* [19]), respectively with the new version of the `WebApp` metamodel.

Figure 2 shows the initial version of the `WebApp` metamodel consisting of modeling constructs for developing Web applications implemented by following the Model-View-Controller pattern (MVC)[14]. The `WebApp` metamodel permits to specify models like the one shown in Fig. 3.a.

Figure 3.b shows different JavaScript (JS) and HTML files automatically generated by means of an Acceleo-based[3] `WebApp2HTML5` transformation applied on the model shown in Fig. 3.a. The generated source code consists of the `index.html` file used as main entry-point of the application and different JS files, one for each main building block

[3] `https://eclipse.org/acceleo/`

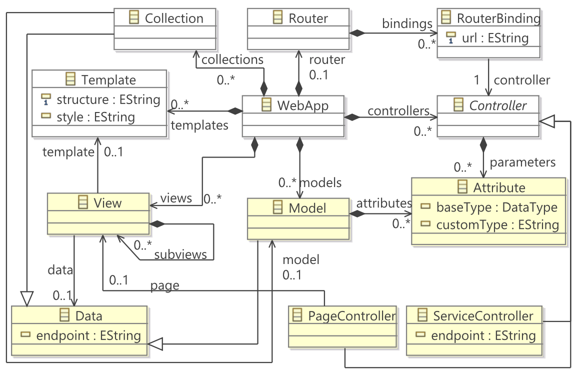
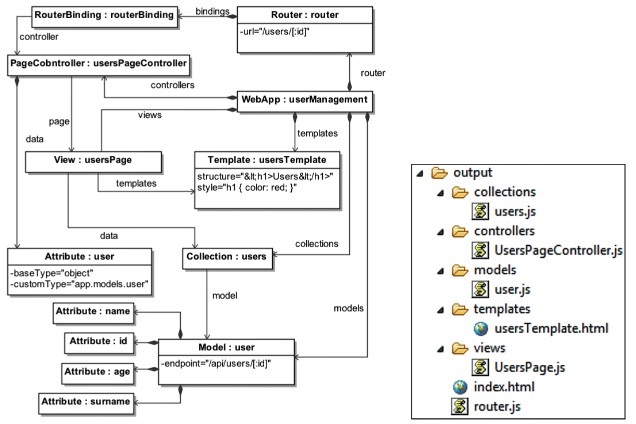


Figure 2. The `WebApp` metamodel

of the application i.e., models, collections, templates, views, the controller and the router. Acceleo transformations are based on templates that identify repetitive and static parts of the applications, and embody specific queries on the source models to fill the dynamic parts.

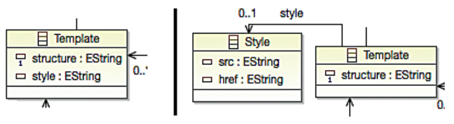
As previously mentioned, metamodels are living entities that can be changed during their life-cycles. Thus the metamodel shown in Figure 2 can be subject to a number of changes that affect all the artifacts defined on it. In the remaining of the paper, we focus on the effects that breaking and unresolvable changes might have on existing Acceleo transformations. In [2] we already proposed a tool-supported approach able to adapt Acceleo-based transformations with respect to the performed changes on the source metamodels. In particular, the approach already automates the management of non-breaking changes and breaking and resolvable ones. Concerning breaking and unresolvable changes the approach applies default heuristics that necessarily have to be checked by developers once the whole adaptation process has been executed.

Even though the availability of a tool which is able to support non-breaking and breaking and resolvable changes represents an important achievement, managing breaking and unresolvable changes by hand still remains an error-prone

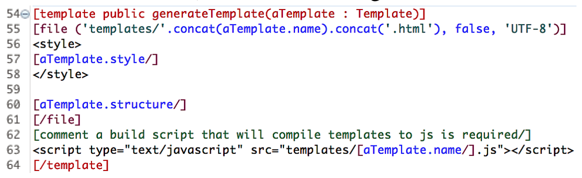


a) WebApp model b) WebApp source code

Figure 3. Simple user management system



a) Metamodel change



```
54 [template public generateTemplate(aTemplate : Template)]
55 [file ('templates/'.concat(aTemplate.name).concat('.html'), false, 'UTF-8')]
56 <style>
57 [aTemplate.style/]
58 </style>
59 
60 [aTemplate.structure/]
61 [/file]
62 [comment a build script that will compile templates to js is required/]
63 <script type="text/javascript" src="templates/[aTemplate.name/].js"></script>
64 [/template]
```

b) Corrupted template

Figure 4. *Extract metaclass* change

process, which necessarily demands a dedicated support. In the remaining of the section we discuss the *extract metaclass* change as a representative metamodel modification that supports such a claim. Because of space limitation, we limit the explanatory discussion to this metamodel change only. A treatment of all the changes included in the catalog that we initially described in [1] and that we are iteratively extending [18] is beyond the scope of this paper.

Figure 4.a shows the new `Style` metaclass with two attributes (*src*, *href* both of type String) that have been added in the evolved version of the `WebApp` metamodel. Such a new metaclass has been added to represent an external style sheet document that can be applied on a given template. The attribute `style` in the old metaclass `Template` has been removed and a new reference with the same name has been added, by giving place to a replacement of an attribute with a reference. Such a change affects the Acceleo template devoted to the management of `Template` elements (see Fig. 4.b). In particular, the `[aTemplate.style]` expression at line 57 cannot be evaluated since the attribute `style` is not existing in the new version of the metaclass `Template`. In order to make the template again applicable, it is possible to adapt the expression in two alternative ways, i.e., `[aTemplate.style.src]` or `[aTemplate.style.href]`, each corresponding to the `EString` attributes in `Style`.

3. Proposed adaptation approach

In this section we propose a *human-in-the-loop* adaptation process (Section 3.1), which is an improvement of the work in [4] and that relies on an extension of the EMFMigrate language [5] presented in Section 3.2. The proposed extension permits users (i.e., humans that have developed the modeling artifacts to be migrated) to provide feedback during the execution of migration programs as discussed in Section 3.3.

3.1 Process overview

Figure 5 is an overview of the proposed adaptation process, which enables user feedback during the execution of migration phases. *Adaptor* is the entry-point of the overall process and given two subsequent versions of the same metamodel, it invokes the operation *getMetamodelChanges* of the *MetamodelComparator* component which gives back the calculated metamodel differences. For each metamodel change properly represented in a difference model, *Adaptor* invokes the *Migrator* component, which in turn involves different components depending on the kinds of changes to be managed. If the change is not breaking, then all the related elements of the artifact to be co-evolved are copied by means of the *ConservativeCopier* object similarly to what the Flock [20] approach does. If the change is breaking and resolvable then the *BRCMigrator* is involved in order to migrate the affected artifacts. Finally, if the change is breaking and unresolvable then the *BUCMigrator* is involved, which in turn properly asks user inputs that are necessary to resolve the change.

In the next section we describe the tool we have implemented to support the development of the `Migrator` and the `UserDialag` components shown in Fig. 5. Since the management of non-breaking, and breaking and resolvable changes has been already discussed elsewhere [1, 5–7], in the next section we focus on the management of breaking and unresolvable changes that represents the main novelty of this paper.

3.2 Supporting tool

The specification of migration programs able to manage breaking and unresolvable changes is performed by means of an extended version of EMFMigrate [23] which is presented in the remaining of the section. Its implementation relying on the Epsilon platform is also discussed.

3.2.1 EMFMigrate in a nutshell

EMFMigrate is a domain specific language for specifying migration programs consisting of migration rules applied on a given artifact *A* conforming to a metamodel *MM*. Migration rules are applied if the corresponding guards evaluated on an input *delta* model hold. Such delta model represents the changes operated on the initial metamodel *MM*. The body of a migration rule consists of a sequence of rewriting rules like the following:

$$s[guard] \rightarrow t_1[assign_1]; t_2[assign_2]; ...; tn[assign_n]$$

where $s, t_1, \ldots, t_n$ refer to metaclasses of the considered artifact metamodel (e.g., the metamodel of Acceleo templates to be migrated), and $guard$ is a boolean expression which has to be *true* in order to rewrite s with t_1, t_2, and t_n. It is possible to specify the values of the target term properties by means of assignment operations (see $assign_i$ above). The guard of each rule is evaluated on the *delta* model in order to apply the corresponding migration rule. The *delta* model can be manually specified, or automatically generated by means of existing model differencing approaches (e.g., EMFCompare[4]). EMFMigrate, originally implemented by a semantic anchoring towards EMFTVM [23], has been suc-

[4] `http://www.eclipse.org/emf/compare`

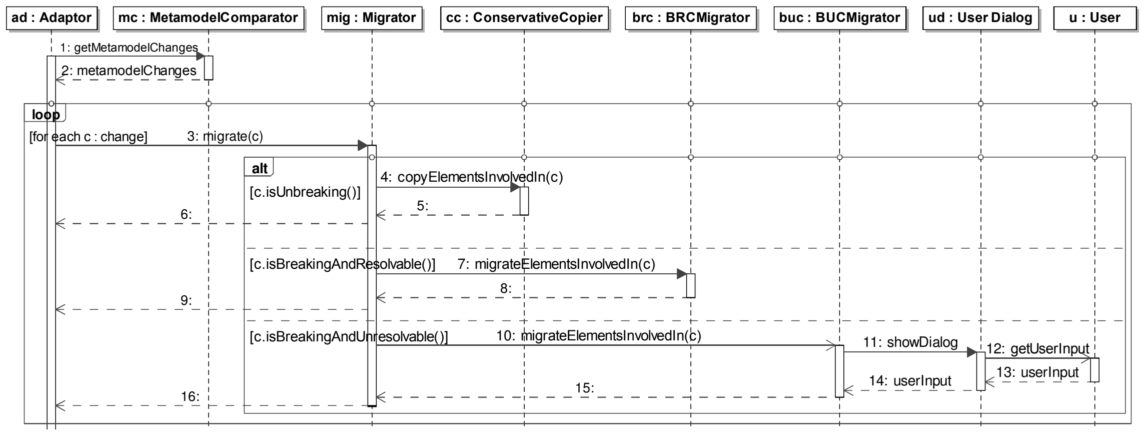


Figure 5. Overview of the *adaptation* flow

cessfully applied to migrate different kinds of artifacts [4], even though before the extension proposed in the following it was not effective in managing breaking and unresolvable changes. In fact, in such cases the language permitted to specify at development time default adaptations by implementing some predefined heuristics.

Listing 1 is a fragment of the `AcceleoMigration` program, which has been developed for adapting Acceleo-based templates like the one discussed in the previous section. In lines 6-15 the rule specifies the metamodel changes that have to match with the content of the `delta` model in order to trigger the subsequent migration rules. The specified changes represent the extract metaclass modification discussed in Section 2. In particular, according to the specified guard, the `extractMetaClass` rule matches when a metaclass `cc1` is changed and a new metaclass `ac2` is added. The changes on the `cc1` consist of the addition of a new reference `ar1` and the deletion of an existing attribute `da1`. According to the `where` expression at line 27, the metaclass `ac2` should be the type of the added reference `ar1` in `cc1`, and the name of `ar1` should be the same of the deleted attribute `da1`. By considering the running example, the guard specified in lines 6-15 matches with the extract metaclass change shown in Fig. 4. In particular, `cc1` matches with the metaclass `Template` of the `WebApp` metamodel, whereas `ac2` matches with the added `Style` metaclass. The reference `style` of the changed `Template` metaclass matches with the `ar1` reference of the guard.

3.2.2 Extension of the EMFMigrate language

As discussed in Sect. 2, when the extract metaclass change occurs there might be different ways to migrate affected Acceleo templates. Instead of choosing one of the possible ways and fixing such ambiguities at development time, we have introduced in EMFMigrate the new construct `prompt` which permits *i)* at *development-time* to specify alternative adaptations for each metamodel change, and *ii)* users to be noticed about them and fix the ambiguities at *run-time*. In the migration program shown in Listing 1 the prompt construct (see line 20) is used to fix all the `PropertyCallExp` elements of existing Acceleo templates that refer to metamodel features that have been affected by the extract metaclass change (e.g., the expression `aTemplate.style` in Fig. 4.b). In the shown migration program, the ambiguity is solved by asking users to choose the added attribute `aa1` or `aa2` that will be instantiated at run-time. In the running example, such attributes would match with the attributes `src` and `href`, respectively of the metaclass `Template`.

Listing 1. Extract metaclass migration for Acceleo templates in extended EMFMigrate

```
 1 migration AcceleoMigration;
 2 metamodel MTL;
 3 migrate AcceleoTransformation:MTL with delta;
 4 ...
 5 rule extractMetaClass [
 6   package cp1 = changePackage{
 7     class cc1 = changeClass {
 8       attribute da1 = deleteAttribute {}
 9       reference ar1 = addReference {}
10     }
11     class ac1 = addClass {
12       attribute aa1 = addAttribute {}
13       attribute aa2 = addAttribute {}
14     }
15   }]
16 {
17   propCallExpOld : MTL!PropertyCallExp -> propCallExpNew
         : MTL!PropertyCallExp [
18     name <- propCallExpNew.name,
19     ...
20     var attrChosen <- System.prompt("Choice the migration
           operation",Sequence{aa1,aa2}),
21     eType <- attrChosen.eType,
22     source <- attrChosem.eContainingClass,
23     source.eType <- ac1,
24     referredProperty <- attrChosen
25     ...];
26 }
27 where [ar1.eType = ac1 and ar1.name = da1.name]...
```

3.2.3 Implementation

As shown in Fig. 6 the implementation of the extended EMFMigrate has been done by means of the `EMFMigrate2-Epsilon` transformation[5] targeting the Epsilon Pattern Language (EPL) and the Epsilon Transformation Language (ETL). EPL implements a pattern matching language, which

[5] `https://github.com/MDEGroup/EMFMigrate`

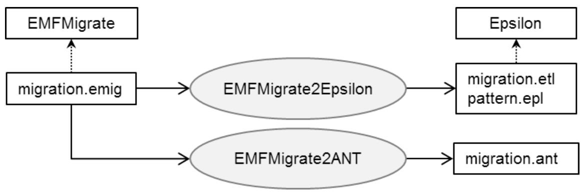


Figure 6. Generation of Epsilon artifacts from EMFMigrate

provides support for specifying patterns that involve elements of models conforming to different modelling technologies. ETL is a hybrid, rule-based model-to-model transformation language built on top of EOL[6]. ETL provides all the standard features of a transformation language, and has the ability to stop the execution of the transformation flow and then ask for user intervention.

Given an EMFMigrate specification like the one shown in Listing 1, the developed `EMFMigrate2Epsilon` transformation generates EPL specifications (e.g., see Listing 2) and ETL transformations consisting of operations and transformation rules (e.g., see Listing 3). More specifically, the guard of each migration rule in EMFMigrate gives place to a corresponding EPL pattern which represents how the specified guard of the migration rule can be matched with elements in the source delta model. The generated pattern plays a key role during the application of the generated ETL transformation, which consists of several transformation rules and operations each devoted to the management of a specific metamodel change. For instance, Listing 3 shows the ETL operation and transformation rule, which have been generated to manage the `extractMetaClass` metamodel change. In particular, the `extractMetaClass` rule makes use of the generated operation `is_extractMetaClass` in order to check if the considered source element has to be migrated because of an operated extract metaclass change. The implementation of the `is_extractMetaClass` operation makes use of the pattern in Listing 2.

Listing 2. Generated EPL pattern specifying the *extract metaclass* change

```
pattern extractMetaClass
 cp1 : Delta!ChangedEPackage,
 cc1 : Delta!ChangedEClass,
 da1 : Delta!DeletedEAttribute,
 ar1 : Delta!AddedEReference,
 ac1 : Delta!AddedEClass,
 aa1 : Delta!AddedEAttribute,
 aa2 : Delta!AddedEAttribute {
 match : cp1.eClassifiers.includes(cc1)
 and cc1.eStructuralFeatures.includes(da1)
 and cc1.eStructuralFeatures.includes(ar1)
 and cp1.eClassifiers.includes(ac1)
 and ac1.eStructuralFeatures.includes(aa1)
 and ac1.eStructuralFeatures.includes(aa2)
}...
```

Listing 3. Generated ETL code related to the `extractMetaClass` migration program

```
operation is_extractMetaClass (propCallExpOld : mtl!
    PropertyCallExp) : Boolean {
var cp1 = Delta!ChangedEPackage.allInstances()->select(c
   | c.instanceOf(Pattern!extractMetaClassCp1));
var cc1 = Delta!ChangedEClass.allInstances()->select(c |
   c.instanceOf(Pattern!extractMetaClassCc1));
var da1 = Delta!DeletedEAttribute.allInstances()->select(
   c | c.instanceOf(Pattern!extractMetaClassDa1));
var ar1 = Delta!AddedEReference.allInstances()->selectOne
   (c | c.instanceOf(Pattern!extractMetaClassAr1));
var ac1 = Delta!AddedEClass.allInstances()->selectOne(c |
    c.instanceOf(Pattern!extractMetaClassAc1));
var aa1 = Delta!AddedEAttribute.allInstances()->select(c
   | c.instanceOf(Pattern!extractMetaClassAa1))->first
   ();
var aa2 = Delta!AddedEAttribute.allInstances()->select(c
   | c.instanceOf(Pattern!extractMetaClassAa2))->first
   ();
return (not Pattern!extractMetaClass.isUndefined()) and
 cc1.applicationElement = propCallExpOld.source.eType
     and
 da1.applicationElement = propCallExpOld.source.
     referredProperty
 and  ac1.name = ar1.eType.name and da1.name = ar1.name;
}
rule extractMetaClass
 transform propCallExpOld : mtl!PropertyCallExp
 to propCallExpNew : mtl!PropCallExp {
  guard: is_extractMetaClass(propCallExpOld)
   var aa1 = Delta!AddedEAttribute.allInstances()->select
        (c | c.instanceOf(Pattern!splitAttributeAa1))->
        first();
   var aa2 = Delta!AddedEAttribute.allInstances()->select
        (c | c.instanceOf(Pattern!splitAttributeAa2))->
        first();
   propCallExpNew.name <- propCallExpOld.name;
   var attrChosen = System.user.choose('Choice␣the␣
        migration␣operation', Sequence{aa1.convert(), aa2
        .convert()});
   propCallExpNew.eType <- attrChosen.eType;
   propCallExpNew.source <- attrChosen.eContainingClass;
   propCallExpNew.referredProperty <- attrChosen;
   ...  }...
```

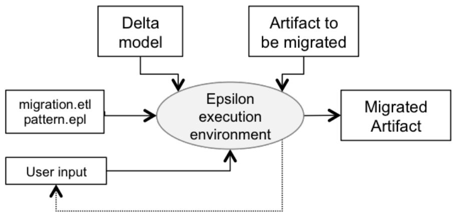
It is important to remark that the generated ETL transformations consist also of rules (not shown in Listing 3) that perform a conservative copy of all those elements that have not been affected by the occurred metamodel changes, and of rules managing breaking and resolvable changes. As discussed in the next section the execution of the generated Epsilon artifacts is performed by means of ANT documents, which are also generated by means of the `EMFMigrate2ANT` transformation shown in Fig. 6.

3.3 Execution of migration programs

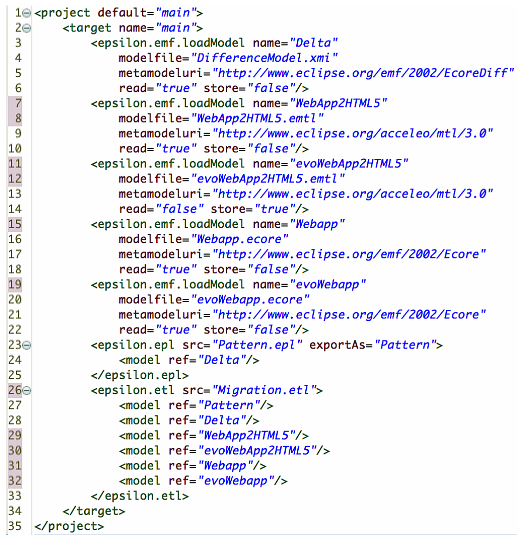
The execution of EMFMigrate migration programs is performed according to the workflow shown in Fig. 7.a. In particular, the generated Epsilon artifacts are executed by means of the Epsilon execution environment by taking as input the artifact to be migrated and the difference model representing the operated metamodel changes. The required user feedback is also asked when needed during the migration that once completed produces the migrated artifacts.

Technically the workflow shown in Fig. 7.a is realized by means of generated ANT documents like the one show in Fig.7.b, which is related to the `WebApp` running example. The ANT document consists of a number of tasks. In particular, by means of the `epsilon.emf.loadModel` task all the required artifacts are loaded i.e., the `Delta` model (see lines 3-6) representing the differences of the subsequent versions of the `WebApp` metamodels loaded in lines 15-22,

[6] http://www.eclipse.org/epsilon/doc/eol/



a) Workflow

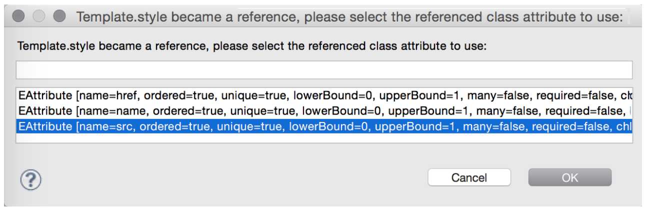


```
1  <project default="main">
2      <target name="main">
3          <epsilon.emf.loadModel name="Delta"
4              modelfile="DifferenceModel.xmi"
5              metamodeluri="http://www.eclipse.org/emf/2002/EcoreDiff"
6              read="true" store="false"/>
7          <epsilon.emf.loadModel name="WebApp2HTML5"
8              modelfile="WebApp2HTML5.emtl"
9              metamodeluri="http://www.eclipse.org/acceleo/mtl/3.0"
10             read="true" store="false"/>
11         <epsilon.emf.loadModel name="evoWebApp2HTML5"
12             modelfile="evoWebApp2HTML5.emtl"
13             metamodeluri="http://www.eclipse.org/acceleo/mtl/3.0"
14             read="false" store="true"/>
15         <epsilon.emf.loadModel name="Webapp"
16             modelfile="Webapp.ecore"
17             metamodeluri="http://www.eclipse.org/emf/2002/Ecore"
18             read="true" store="false"/>
19         <epsilon.emf.loadModel name="evoWebapp"
20             modelfile="evoWebapp.ecore"
21             metamodeluri="http://www.eclipse.org/emf/2002/Ecore"
22             read="true" store="false"/>
23         <epsilon.epl src="Pattern.epl" exportAs="Pattern">
24             <model ref="Delta"/>
25         </epsilon.epl>
26         <epsilon.etl src="Migration.etl">
27             <model ref="Pattern"/>
28             <model ref="Delta"/>
29             <model ref="WebApp2HTML5"/>
30             <model ref="evoWebApp2HTML5"/>
31             <model ref="Webapp"/>
32             <model ref="evoWebapp"/>
33         </epsilon.etl>
34     </target>
35 </project>
```

b) Simple ANT script

Figure 7. Execution of migration programs

the `WebApp2HTML5` Acceleo transformation to be adapted (lines 7-10), the generated EPL document (lines 23-25), and the generated ETL transformation (lines 26-33). The execution of the ETL transformation generates the migrated Acceleo transformation, which is stored in the model specified in lines 11-14.



Template.style became a reference, please select the referenced class attribute to use:

Template.style became a reference, please select the referenced class attribute to use:

```
EAttribute [name=href, ordered=true, unique=true, lowerBound=0, upperBound=1, many=false, required=false, cl
EAttribute [name=name, ordered=true, unique=true, lowerBound=0, upperBound=1, many=false, required=false,
EAttribute [name=src, ordered=true, unique=true, lowerBound=0, upperBound=1, many=false, required=false, chl
```

Figure 8. Generated user dialog related to the *Extract metaclass* change

Interestingly, during the execution of migration programs user inputs are properly asked as shown in Fig 8 which shows real user dialogs that are related to the management of the breaking and unresolvable changes discussed in Section 2. The generated prompt shown in Fig. 8, permits solving ambiguities related to the new reference `style`. The dialog shows all the possible attributes included in the new referenced `Style` metaclass that can be used to fix the problem.

4. Related work

Over the last years, the coupled evolution problem has been carefully investigated. In [1, 20] authors proposed an approach to deal with the coupled evolution of metamodels and models. In [11] Garcia et al. defined the correspondences between changes applied on metamodels and how they affect existing model transformations. In [22] authors propose a transformational approach to assist metamodel evolution by stepwise adaptation. They defined several relations between metamodels to characterize metamodel evolution. In [2] we have presented a similar approach for dealing with metamodel changes and Acceleo template-based code generators, without considering breaking and unresolvable changes.

In [17] authors use graph transformations to support model co-evolution. In particular, co-evolutions are specified as related graph transformations ensuring well-formed model migration results. The model migration approach has been extended by allowing transformation rules that have less restrictions so that graph manipulations such as merging of types and retyping of graph elements are allowed [16]. In [15] authors presented an approach devoted to an evolution method for model transformations in the context of GReAT [12]. Based on the evolution, the approach is able to automatically migrate certain parts of the transformations. When automation is not possible, the algorithms automatically warn the user if some semantic information is missing. The lack of information is manually fixed by users after the execution of the automatic part of the interpreter evolution. All the previously mentioned approaches aim at automatically migrate artifacts because of operated metamodel changes and when this is not possible they provide some support to the user when the automated process has completed and manual intervention is still required. Such approaches differ to what has been presented in this paper since they do not provide the means to guide users at run-time during the execution of the migration processes.

In [21] the authors present an interactive and iterative approach to meta-model construction enabling the specification of model fragments where they say that if a change is unresolvable, the user is asked to provide additional information or to discard the no longer conformant fragment. Even though this can appear similar to what has been presented in this paper, the approach proposed in [21] is intended to be used during metamodeling phases and to deal with the metamodel and model coupled evolution problem only. Our approach aims at dealing with the coupled evolution problem occurring when metamodels are already in production and several related artifacts have been developed from them. Edapt [8] is a promising tool providing mechanisms to record the changes that are executed on Ecore models. The approach permits to attach migration instructions written in Java to metamodel changes in order to restore the affected models. Differently to Edapt, EMFMigrate is a rule-based DSL enabling users to specify migration actions that are translated to a target transformation language.

5. Conclusions and future work

In this paper we addressed the problem of supporting users to manage the adaptation of artifacts which have been affected by breaking and unresolvable metamodel changes. A general adaptation process has been proposed by making explicit those activities that are related to the management of user feedback. So far breaking and unresolvable changes have been solved by deciding at development time how to solve possible ambiguous situations, or by partially adapting affected artifacts. Thus after the migration steps, usually users have to perform error-prone and time-consuming activities to completely adapt the artifacts that have been only partially adapted by the employed migration approach. With the techniques proposed in this paper, we permit to involve users during the execution of the migration phases and to fix ambiguities at run-time. The proposed process is supported by an extension of the EMFMigrate language, which is implemented atop of the Epsilon framework. In the future, we plan to perform an extensive evaluation of the approach by undertaking different actions. In particular, we plan to consider all the metamodel changes reported in the catalog we are maintaining [18]. In turn, we will apply the approach to breaking and resolvable changes related to other kinds of coupled evolution problems e.g, those involving models, model-to-model transformations, and editors. We will also investigate how to extend the approach to migrate multiple artifacts defined on the same metamodel.

References

[1] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Procs.of EDOC 2008*, pages 222–231. IEEE Computer Society, 2008.

[2] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio. Dealing with the coupled evolution of metamodels and model-to-text transformations. In *ME'14 at MoDELS 2014*, 2014.

[3] D. Di Ruscio, R. Lämmel, and A. Pierantonio. Co-evolution of GMF Editor Models. In *Procs. SLE'11*, volume 6563 of *LNCS*, pages 143–162. Springer, 2011.

[4] D. Di Ruscio, L. Iovino, and A. Pierantonio. Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems. In *Procs. of ICGT'12*, pages 20–37. Springer-Verlag, 2012.

[5] D. Di Ruscio, L. Iovino, and A. Pierantonio. Coupled evolution in model-driven engineering. *Software, IEEE*, 29(6): 78–84, Nov 2012.

[6] D. Di Ruscio, L. Iovino, and A. Pierantonio. A methodological approach for the coupled evolution of metamodels and atl transformations. In *Procs. ICMT'13*, volume 7909 of *LNCS*, pages 60–75. Springer, 2013.

[7] D. Di Ruscio, L. Iovino, and A. Pierantonio. Managing the coupled evolution of metamodels and textual concrete syntax specifications. In *SEAA'13*, pages 114–121, Sept 2013.

[8] Eclipse.org. Edapt Migrating EMF Models. `http://www.eclipse.org/edapt`.

[9] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. A model-driven approach to automate the propagation of changes among architecture description languages. *Software & Systems Modeling*, 11(1):29–53, 2012.

[10] G. Gabrysiak, H. Giese, A. Lüders, and A. Seibel. How Can Metamodels Be Used Flexibly? In *Proc. of ICSE 2011 Workshop on Flexible Modeling Tools*, 2011.

[11] J. Garca, O. Diaz, and M. Azanza. Model transformation co-evolution: A semi-automatic approach. In *Procs. SLE'13*, volume 7745 of *LNCS*, pages 144–163. Springer, 2013.

[12] G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the use of graph transformation in the formal specification of model interpreters. *J. UCS*, 9(11):1296–1321, 2003.

[13] A. Kusel, J. Etzlstorfer, E. Kapsammer, P. Lange, W. Retschitzegger, J. Schoenboeck, W. Schwinger, and M. Wimmer. A Systematic Taxonomy of Metamodel Evolution Impacts on OCL Expressions. In *ME'14 at MoDELS 2014*, Sept. 2014.

[14] A. Leff and J. Rayfield. Web-application development using the model/view/controller design pattern. In *EDOC '01*, pages 118–127, 2001.

[15] T. Levendovszky, D. Balasubramanian, A. Narayanan, and G. Karsai. A Novel Approach to Semi-automated Evolution of DSML Model Transformation. In *Procs. SLE'10*, volume 5969 of *LNCS*, pages 23–41. Springer, 2010.

[16] F. Mantz, Y. Lamo, and G. Taentzer. Co-transformation of type and instance graphs supporting merging of types with retyping. *ECEASST*, (61), 2013.

[17] F. Mantz, G. Taentzer, and Y. Lamo. Customizing model migrations by rule schemes. In *Procs. IWPSE'13*, pages 1–10. ACM, 2013.

[18] MDE Research Group. The Metamodel Refactorings Catalog. `http://www.metamodelrefactoring.org`. University of L'Aquila.

[19] D. Mendez, A. Etien, A. Muller, and R. Casallas. Towards Transformation Migration After Metamodel Evolution. In *Model and Evolution Workshop*, Olso, Norway, Oct. 2010.

[20] L. M. Rose, D. S. Kolovos, R. F. Paige, F. A. C. Polack, and S. M. Poulding. Epsilon flock: a model migration language. *Software and System Modeling*, 13(2):735–755, 2014.

[21] J. Sánchez-Cuadrado, J. Lara, and E. Guerra. Bottom-up meta-modelling: An interactive approach. In *MODELS*, volume 7590 of *LNCS*, pages 3–19. Springer, 2012.

[22] G. Wachsmuth. Metamodel Adaptation and Model Co-adaptation. In E. Ernst, editor, *ECOOP 2007 Object-Oriented Programming*, volume 4609 of *LNCS*, pages 600–624. Springer, 2007.

[23] D. Wagelaar, L. Iovino, D. Di Ruscio, and A. Pierantonio. Translational Semantics of a Co-evolution Specific Language with the EMF Transformation Virtual Machine. In *Procs. ICMT 2012*, volume 7307 of *LNCS*, pages 192–207. Springer, 2012.