Traceability Visualization in Metamodel Change Impact Detection

Juri Di Rocco
University of L'Aquila
I-67100 L'Aquila, Italy
juri.dirocco@univaq.it

Davide Di Ruscio
University of L'Aquila
I-67100 L'Aquila, Italy
davide.diruscio@univaq.it

Ludovico Iovino
University of L'Aquila
I-67100 L'Aquila, Italy
ludovico.iovino@univaq.it

Alfonso Pierantonio
University of L'Aquila
I-67100 L'Aquila, Italy
alfonso.pierantonio@univaq.it

ABSTRACT

In Model-Driven Engineering metamodels are typically at the core of an *ecosystem* of artifacts assembled for a shared purpose. Therefore, modifying a metamodel requires care and skill as it might compromise the integrity of the ecosystem. Any change in the metamodel cannot prescind from recovering the ecosystem validity. However this has been proven to be intrinsically difficult, error-prone, and labour-intensive. This paper discusses how to generate and visualize traceability information about the dependencies between artifacts in a ecosystem and their related metamodel. Being able to understand how and where changes affect the ecosystem by means of intuitive and straightforward visualization techniques can help the modeler in deciding whether the changes are sustainable or not at an early stage of the modification process.

1. INTRODUCTION

Model-Driven Engineering [1] (MDE) aims at capturing *problems* in terms of concepts that are much closer to the application domain rather than to the underlying technological assets. Problems are then consistently mapped to *solutions* by means of model operations generally defined by model transformations. Both problems and solutions are described with the help of *models* expressed in terms of concepts and relationships among them given in *metamodels*. Additionally metamodels characterize also a wide range of further artifacts, including model transformations, textual and diagrammatic editors, and code generators which are *typed*[1] after them. Therefore, metamodels are at the core of an aggregation of artifacts assembled for a shared purpose, called *metamodeling ecosystem* [2, 3].

Similarly to software, metamodels can be subject to internal or external evolutionary pressures [4]. However, because of the dependencies among the artifacts, changing a metamodel requires the rest of the ecosystem to be adapted in order to remain valid or consistent. As a consequence, before changing a metamodel it is of crucial relevance to measure the impact of the changes among the artifacts in order to understand whether the evolution is sustainable or not. In a previous work [5] we have already presented techniques based on simple cost functions which may help in understand how much changing a metamodel is affordable. Nonetheless, quantitatively analyzing the impact can be misleading as it does not convey any particular about *where* and *how* the adaptations have to be applied.

In this paper, we present a change impact visualization which complements the aforementioned work. In particular, traceability information about the dependencies between the metamodel and the specific artifact under study is generated and visualized. This permits the modeler to assess the significance of the change by using an adaptation density concept which visualizes and gives an *intuition* about how localized and how many adaptations are required within a given artifact fragment. The approach is generic and can be used for different kinds of artifacts.

The paper is structured as follows. Section 2 presents the background. Section 3 outlines a methodology to deal with the co-evolution problem. In Section 4 the visualization of the traceability is explained. Section 5 analyses different results obtained with our approach presenting possible visualizations and their meaning. In Section 6 we discuss alternative approaches and application using TraceVis and analize other approaches to change impact detection and traceability. Section 7 concludes the paper outlining future works.

2. BACKGROUND: COUPLED EVOLUTION PROBLEM

As already mentioned, metamodels rarely exist in isolation since they are the cornerstone upon which many modeling artifacts are built. The problem of how to let the metamodel

[1]For instance, a transformation usually have a source and a target metamodel.

GMLD '13, July 01 2013, Montpellier, France

http://dx.doi.org/10.1145/2489820.2489824.

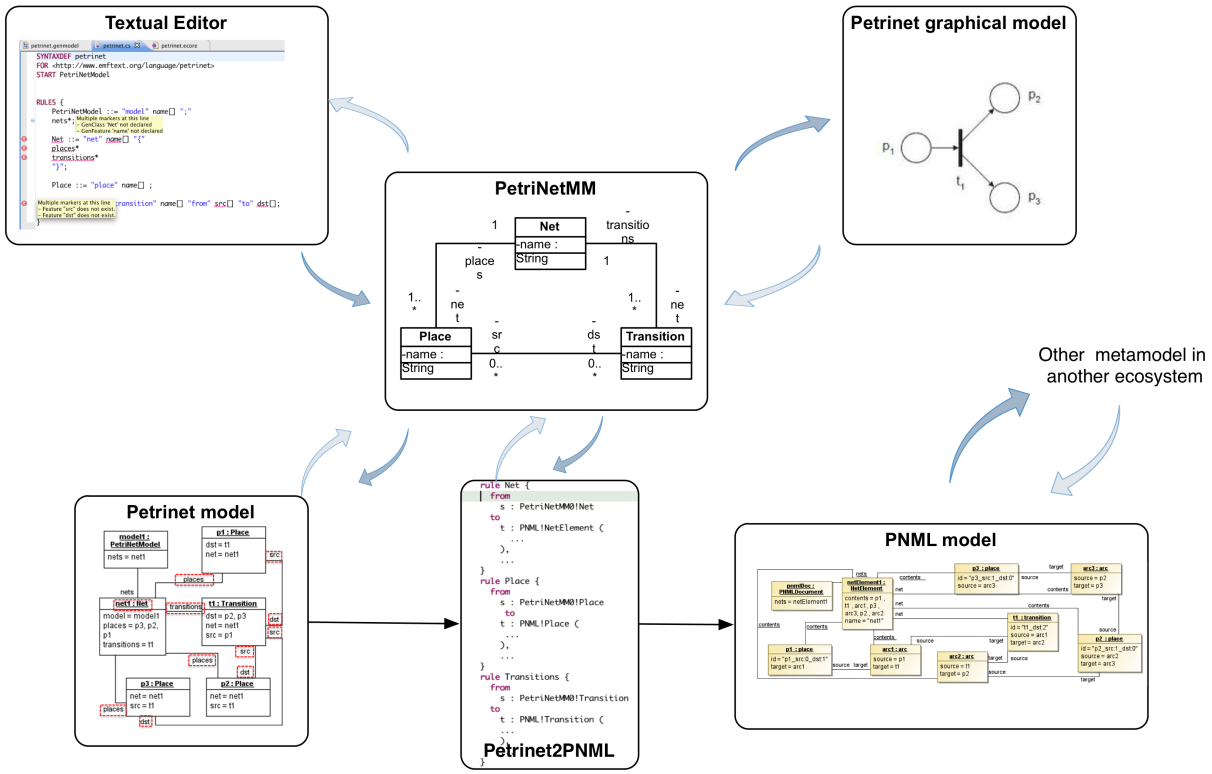

Figure 1: MDE ecosystems

evolve together with the ecosystem has been partially addressed and discussed in [3], while most of the approaches were dealing with the metamodel/model co-evolution (e.g. [4, 6, 7]) and only few work with metamodel/transformation and other artifact co-evolution (e.g., [8, 9]). To better understand the complexity of the problem let us consider the ecosystem represented in Figure 1. The *PetriNetMM* metamodel is given in the middle, PetriNet *models* can be translated by means of *transformations*, and finally different kinds of *editors* permit to enter and manipulate the models.

The metamodel changes can be classified according to the impact on the artifacts. In particular, there are changes that *a)* do not affect the artifacts at all and no adaptation is required; *b)* affect the artifact which can be automatically adapted; and finally *c)* affect the artifacts which cannot be automatically adapted. Furthermore, the adaptations which are required to recover modeling artifacts with respect to the changes executed on the corresponding metamodel, depend on the relation which couple the modeling artifact and the metamodel [2]. An example of metamodel evolution is shown in Figure 2, where the initial version of the PetriNet metamodel in Figure 2 (top) is modified by leading to the version shown at Figure 2 (bottom). Clearly, the validity and consistency of the ecosystem might be compromised because of the corruption of the relations between artifacts and the metamodel [10].

Therefore, it is of paramount importance to be able to trace and visualize the dependencies within the ecosystem in order to enable the modeler to detect those modifications that compromise existing artifacts and especially those that require more labour to be repaired. In the next section, a methodology is discussed to support the adaptation of those artifacts which have been affected by metamodel changes.

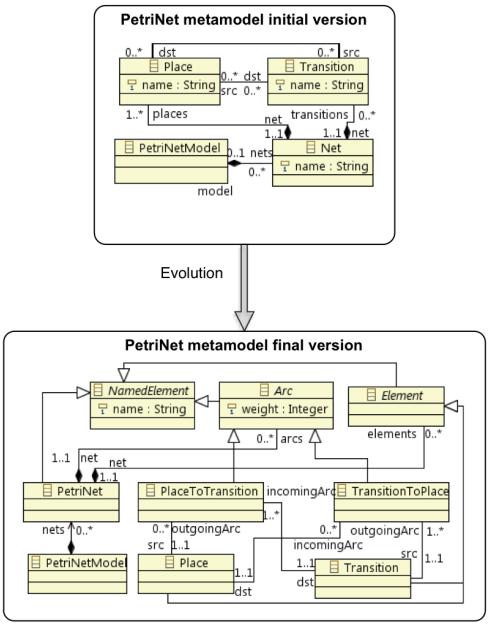

Figure 2: PetriNet metamodel evolution

3. COUPLED EVOLUTION MANAGEMENT PROCESS

If the adaptation is applied with spontaneous or individual skills can give place to inconsistencies between the metamodel and the related artifacts [4]. Therefore, a methodology which prevents the consequent information erosion and provides the modeler with a consistent guidance in the adaptation process can be significant. In Figure 3, a process consisting of a number of activities is presented. It encompasses the specification of the metamodel changes, the evaluation of their impact on the existing artifacts, the sustainability of the induced adaptations, and finally the actual migrations of the affected artifacts. This represents an enhancement of the methodology presented in [5], which did not consider the *change impact visualization* activity. In the remaining of the section, all the activities shown in the figure are discussed separately.

1. *Relation Definition*: this activity is performed only once per each kind of artifact to be adapted. For instance, in the case of ATL transformations [11], the ATL and the ECore metamodels are considered in order to establish correspondences between them. Such correspondences are used later in the process to automatically derive the dependencies between an evolving metamodel and the existing transformations. This activity can be done by using the work in [2] that exploits weaving models and megamodels to specify and manipulate correspondences among related and evolving artifacts. The upper side of Figure 4 shows a sample weaving model (as defined in [2]), which specifies the relation between the ECore metaclass `EClass` and the ATL metaclass `OclModelElement`.

2. *Dependencies Elicitation*: after the previous activity, the definition of the relations can be used to automatically derive the dependencies between the evolving metamodel and the linked artifacts. Such dependencies can be expressed as a weaving model like the one shown in the lower side of Figure 4. It shows the dependencies between the metaclasses of the `PetriNet` metamodel and the elements of a given ATL transformation having it as source metamodel. For instance, the first rule of the transformation shown on the right-hand side of Figure 4 contains an `OclModelElement` named `Net`, thus it is connected with the `EClass` element similarly named on the left-hand side of the figure. Such a dependency link specifies that changing the name of the `Net` metaclass in the `PetriNet` metamodel implies to propagate such a change to each `OclModelElement` linked to it.

3. *Metamodel Changes Specification*: the changes that the modeler wants to apply on a given metamodel should be properly represented in order to enable automatic manipulations and automate subsequent phases of the process. For instance, in this phase it is possible to adopt the metamodel independent approach to difference representation proposed in [12] already used to deal with other coupled evolution problems (e.g., adaptation of models [7], and GMF editors [9]), but other approaches can be used as well.

4. *Change Impact Visualization*: in this phase the elements of the artifact, which has been affected by the specified metamodel changes are graphically shown. In this way, modelers can have a preliminary assessment of the impact that the metamodel changes being applied, will have on the existing artifacts. They can decide to amend such changes, or to go ahead in the process with a more accurate evaluation of the required adaptation cost [5].

5. *Change Impact Analysis / Evaluation Cost*: in this activity the evaluation performed in the previous step is enhanced by considering an appropriate cost function related to the actions, which are required to adapt the affected artifacts [5]. In the specific case of ATL model transformations, according to the dependencies previously elicited, all the transformation elements which are in relation with the changed metamodel elements are identified and used as input for evaluating the adaptation cost. In particular, by considering the affected elements modelers evaluate the cost for adapting the affected transformations. In this respect, if the adaptation is too expensive (according to an established threshold) modelers can decide to refine the metamodel changes to reduce the corresponding costs, otherwise they can accept the applied metamodel changes. The evaluation is based on an adaptation cost function defined separately [5].

6. *Metamodel Changes Commit*: once the metamodel changes have been evaluated, modelers can commit them in order to concretely apply the previously evaluated artifact adaptations.

7. *Artifact Adaptation*: in this activity the existing artifacts which have been affected by the committed metamodel changes are adapted. Proper tools are required to support this step. Over the last years different approaches have been proposed to support the coupled evolution of metamodels and related artifacts. EMFMigrate [13] is one of those tools that presents the advantage of dealing with the diversity of artifacts with a unique notation.

In the next section we focus on the *Change Impact Visualization* activity, since the detection phase [2] and the migration activity [5] have been already treated. We propose a tool chain able to generate a graphical representation of the change impact out of a model representing the differences between two subsequent versions of the same metamodel.

4. CHANGE IMPACT VISUALIZATION

Over the last years, different attempts have been proposed to deal with the problem of supporting the graphical representation of inter-model relations. For instance, in some cases there is the need for representing different viewpoints of a system, or multiple views of it, each capturing a concern of interest for a particular stakeholder. In this direction, the TraceVis tool [14] has been proposed to graphically represent traceability information between structured data. Interestingly, TraceVis has been adopted to visualize traceability links between the source and target models of an executed model transformation [15].

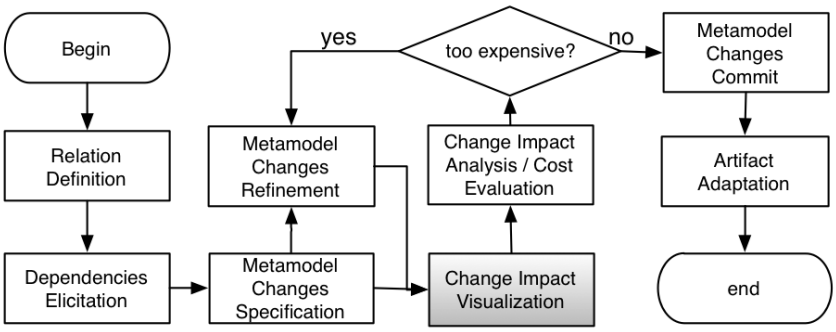

Figure 3: Methodology activities

In this section, we propose a tool chain that aims at using the TraceVis tool to graphically represent the impact that the changes being addressed on a given metamodel can have on existing artifacts. Figure 5 depicts a screenshot of the TraceVis tool at work while representing the dependencies (generated by the tool chain explained later in the section) among three different models. In particular, the part of the figure labelled ② represents the initial version of the PetriNet metamodel shown in Figure 2.a. The part labelled ① represents the difference model (also called delta model hereafter) encoding the differences to be applied on the initial version of the metamodel in order to obtain that in Figure 2.b. The part labelled ③ represents an existing ATL transformation, which has ② as source metamodel, and is affected by the metamodel changes specified in ①. The lines between ① and ② represents the relations between the differences represented in ① and the metamodel elements in ② that are changed. For instance, the `ChangedClass` modification represented in ① is related to the class `Net` because of the renaming operation that has to be performed in order to obtain the new version of the mataclass named as `PetriNet`. The links between ② and ③ represent the impact that changing the `Net` metaclass has on the existing ATL transformation *PetriNet2PNML*[2]. For instance, the `OCLModelElement Net` in the source input pattern of the transformation is linked to the evolving metaclass and indicates a possible issue in case of changing that metaclass `Net`. Of course the package containing the changed class is involved too (see the connection between the package `PetriNet` and the `OCLModel PetrinetMM0` in the transformation). To better understand the connection between ② and ③, inexpert ATL developers can consider Figure 6 that explicitly represents the considered ATL transformation (see the left-hand side of the figure) and its representation in the TraceVis tool (see the right-hand side of the figure). For instance, the input and output models of the transformation header (see `OUT` and `IN` of the second line of the transformation) are the last two elements in the TraceVis representation.

In the remaining of the section we present the proposed tool chain (see Section 4.1) able to generate TraceVis specifications like the one in Figure 5, starting from a dependency model like the one in the lower-hand side of Figure 4, and a difference model representing the metamodel changes being applied. The realization of the tool chain required the implementation of model-to-model, and model-to-code transformations, which are detailed in Section 4.2, and Section 4.3, respectively.

4.1 Proposed tool chain

In order to support the TraceVis visualization of change impacts, we have implemented a tool chain able to generate artefacts that TraceVis is able to open. In particular, TraceVis is able to manage XML documents like the one in Listing 1, which is a fragment of the XML specification of the visualization depicted in Figure 5. Essentially, the document consists of stages, each represent one of the related models. For instance, Listing 1 contains 3 stages (see lines 4, 23, 34) corresponding to the difference model, the PetriNet metamodel, and the affected ATL transformation shown in Figure 5, respectively. Each stage contains the specification of nodes, each representing one of the elements related to one or more elements contained in another model. For instance, the node `75` in line `5` of Listing 1 represents the element of the difference model encoding the metaclass renaming. The attributes that can be specified for each node include the name, the parent node, and additional custom attributes that might be useful for automatic manipulations. After the specification of stages, and consequently the contained nodes, inter-model relations are given (see lines 54-72 in Listing 1).

Listing 1: Fragment of the TraceVis XML file of visualization in Figure 5

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2  <data>
3   <stages>
4     <stage id="1" name="delta1">
5      <node id="75">
6       <attribute name="parent" type="">
7        <modification>76</modification>
8       </attribute>
9       <attribute name="name" type="string">
10       <modification>DELTA_ChangedeClass_Net</
             modification>
11      </attribute>
12     </node>
13     <node id="76">
14      <attribute name="parent" type="">
15       <modification>5</modification>
16      </attribute>
17      <attribute name="name" type="string">
```

[2]This transformation is available from `http://www.eclipse.org/atl/atlTransformations/Grafcet2PetriNet/ExampleGrafcet2PetriNet\[v00.01\].pdf`

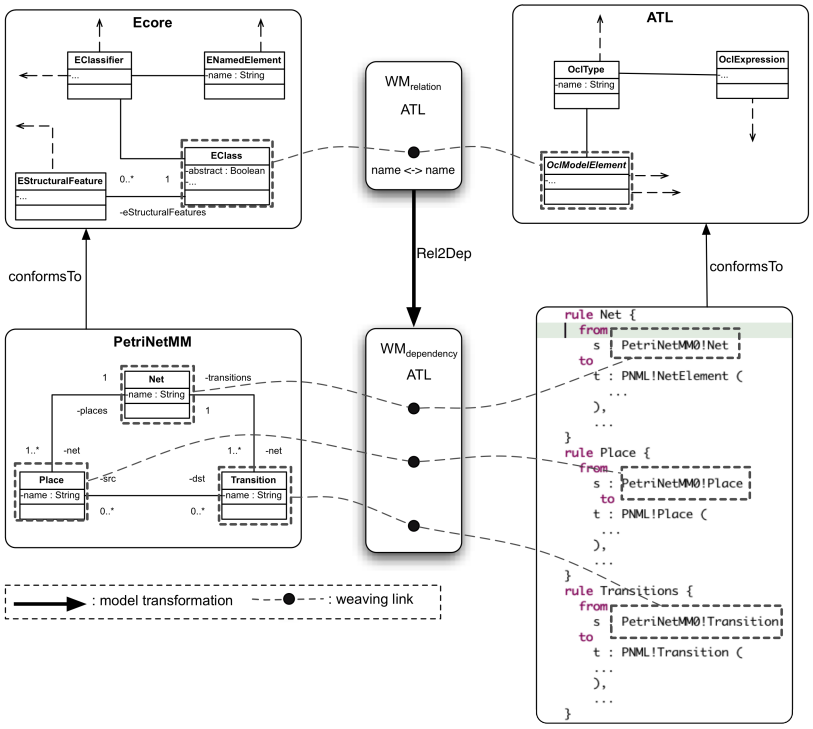

Figure 4: Relation Definition and Dependencies Elicitation

```
      <modification>DELTA_ChangedePackage_petrinet</
          modification>
     </attribute>
    </node>
    ...
   </stage>
   <stage id="2" name="petrinet">
    <node id="23">
     <attribute name="parent" type="">
      <modification>26</modification>
     </attribute>
     <attribute name="name" type="string">
      <modification>MM_eClass_Net</modification>
     </attribute>
    </node>
    ...
   </stage>
   <stage id="3" name="petrinet2GM-ATL-0.2">
    <node id="62">
     <attribute name="parent" type="">
      <modification>37</modification>
     </attribute>
     <attribute name="name" type="string">
      <modification>ATLATL_OCL_MODEL_ELEMENT_Net</
          modification>
     </attribute>
    </node>
    <node id="63">
     <attribute name="parent" type="">
      <modification>71</modification>
     </attribute>
     <attribute name="name" type="string">
      <modification>ATLATL_OCL_MODEL_ELEMENT_Net</
          modification>
     </attribute>
    </node>
    ...
   </stage>
  </stages>
  <relations>
   <relation type="Impact_On">
     <source stage="1">75</source>
     <target stage="2">23</target>
   </relation>
   <relation type="Impact_On">
     <source stage="1">76</source>
     <target stage="2">26</target>
   </relation>
   <relation type="Impact_On">
    <source stage="2">23</source>
    <target stage="3">62</target>
   </relation>
   <relation type="Impact_On">
    <source stage="2">23</source>
    <target stage="3">63</target>
   </relation>
   ...
  </relations>
</data>
```

As shown in Figure 7 the generation of TraceVis XML specifications is performed in different steps. In particular, given two subsequent versions of the same metamodel `MM`, and `MM'`, their differences are calculated by means of the EMFCompare [16]. The outcome of the comparison is the input of the model transformation `EMFCompare2EcoreDiff` [17] able to generate a difference model according to the difference representation approach proposed in [12]. Such a difference model, together with the initial metamodel `MM`, the existing `Artifact` which depends on it, and the `DependencyModel` (defined in the *Dependencies Elicitation* step in Figure 3), are taken as input by the model-to-model transformation `TraceVisGen`. The generated `TraceVisModel` is transformed by means of the model-to-code transformation `TraceVisMM2XML` able to generate a corresponding XML document like the one in Listing 1.

In the next section the `TraceVisGen` transformation is presented, whereas the model-to-code transformation `TraceVisMM2XML` is discussed in Section 4.3.

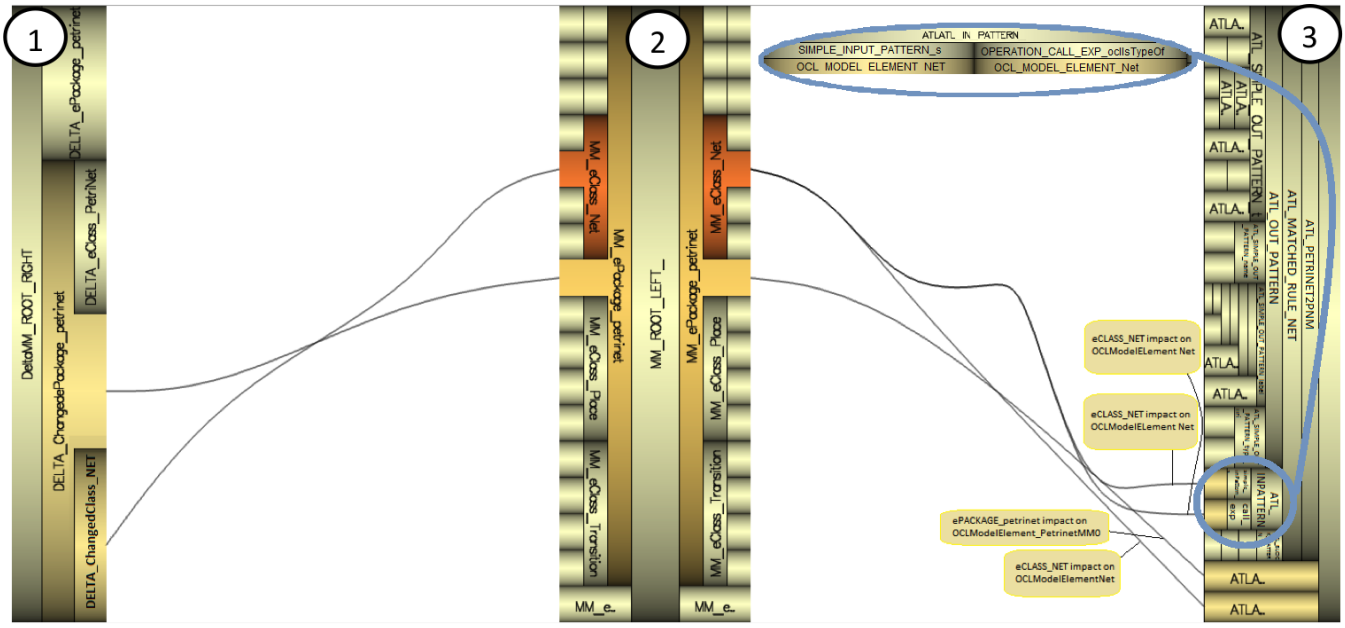

Figure 5: Metaclass renaming visualization

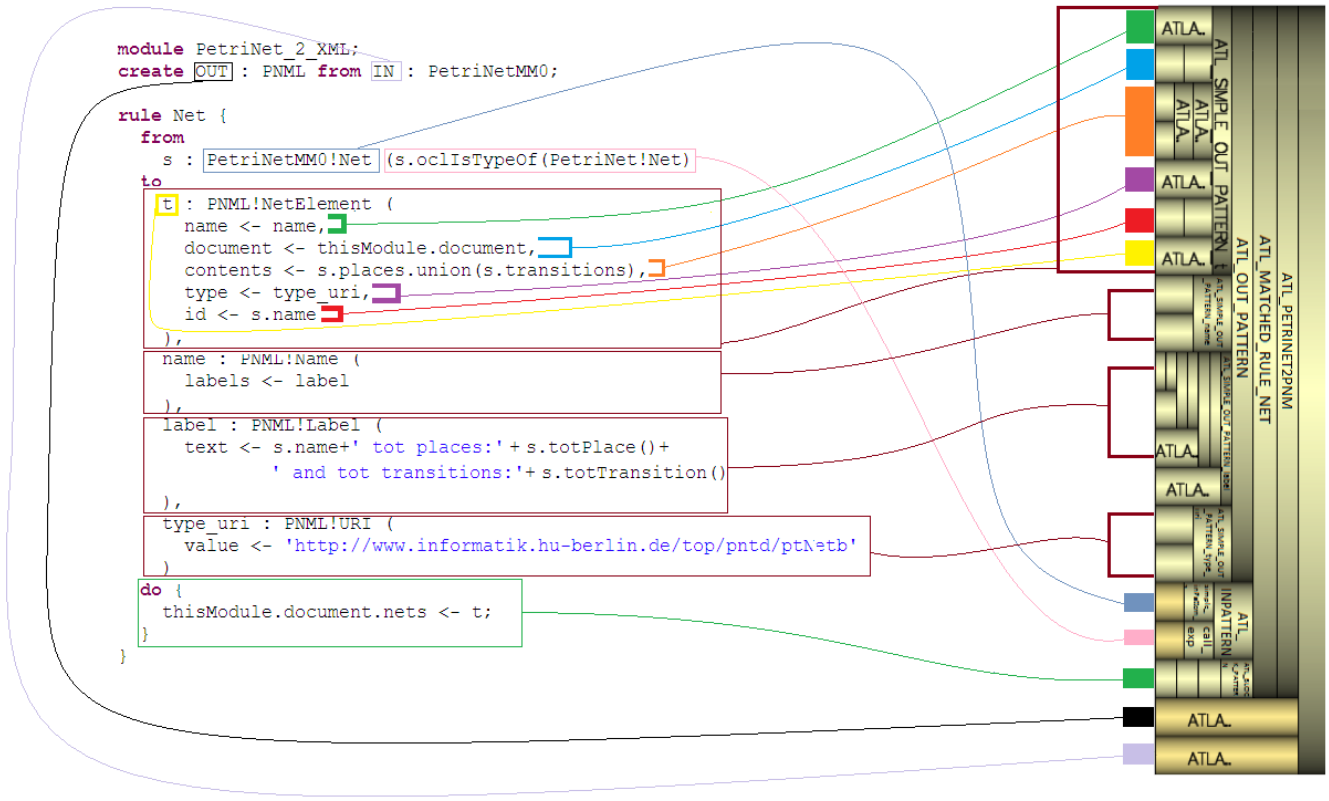

Figure 6: ATL transformation and its representation in TraceVis

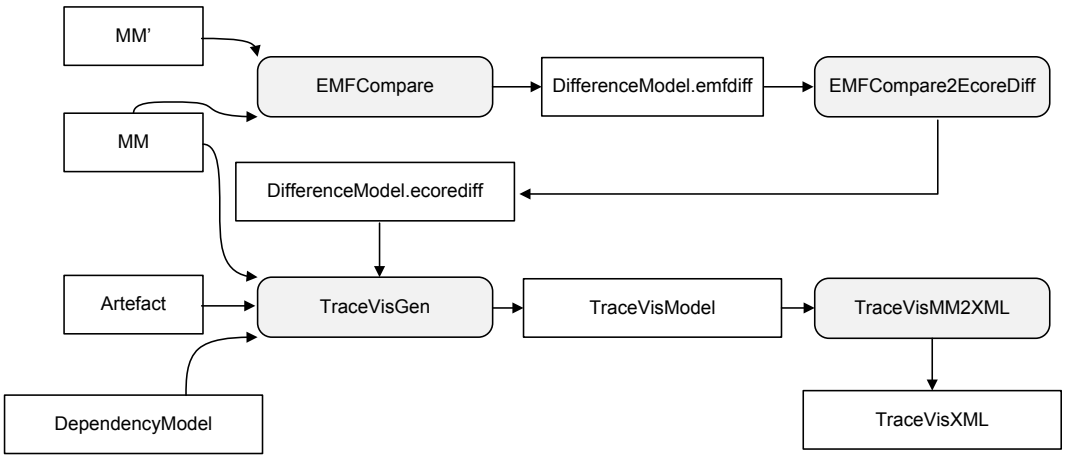

Figure 7: Generation of TraceVis XML specifications

4.2 From dependency models to TraceVis specifications

Splitting the generation of the final XML artifact in two steps resulted to be more simple than having only one model-to-text transformation. In fact, the semantic gap between the two levels of abstraction is too large, and this motivated splitting the generation process in two steps. Thus, the TraceVis metamodel in Figure 8 has been defined in order to enable the generation of intermediate TraceVis models as shown in Figure 7. The generation of such models is performed by means of the ATL transformation shown in Listing 2.

The generation of `Node`, `Attribute` and `Modification` elements is performed by the rule `NodeClass` in lines 4-24. The *conditional* statement in lines 17-21 contains the invocation of `refImmediateComposite` to retrieve the container of an element, and then makes use of the `resolveTemp` function to get the target model that will be generated by an ATL rule combined with a given source model element.

Listing 2: Fragment of the TraceVisGen transformation

```
1 module Adapter;
2 create OUT : TRACEVIS from wmdep : WeavingModel, MMD
      : ECORE, atl : MM;
3
4 rule NodeClass {
5  from
6   s : ECORE!EClass
7  to
8   node: TRACEVIS!Node(
9    id <- thisModule.ID,
10   attribute <- Sequence{attr,attrName}
11  ),
12  attr: TRACEVIS!Attribute(
13   modification <- modi,
14   name <- 'parent'
15  ),
16  modi: TRACEVIS!Modification(
17   parent <- if(s.refImmediateComposite()=
         OclUndefined)
18       then thisModule.leftRoot
19       else
20       thisModule.resolveTemp(s.refImmediateComposite
           (),'node')
21       endif
22  ),
23  ...
24 }
25 ...
26 rule NodeChangedClassDiff {
27  from
28   s : EcoreDiff!ChangedEClass
29  to
30
31  --Relation from inter-model reference
       ApplicationElement
32   rel : TRACEVIS!Relation(
33    source <- sour,
34    target <- targ
35   ),
36   sour: TRACEVIS!Source(
37    node <- thisModule.resolveTemp(s,'node'),
38    stage <- thisModule.MarkstageDelta
39   ),
40   targ: TRACEVIS!Target(
41     node <- thisModule.resolveTemp(s.
          applicationElement,'node'),
42     stage <- thisModule.MarkstageL
43   )
44 }
45 ...
46 rule RelationLink{
47  from
48   s : WeavingModel!Link
49  to
50   node : TRACEVIS!Relation(
51    source <- sour,
52    target <- targ
53   ),
54   sour: TRACEVIS!Source(
55    node <- s.left.getReferredElement(),
```

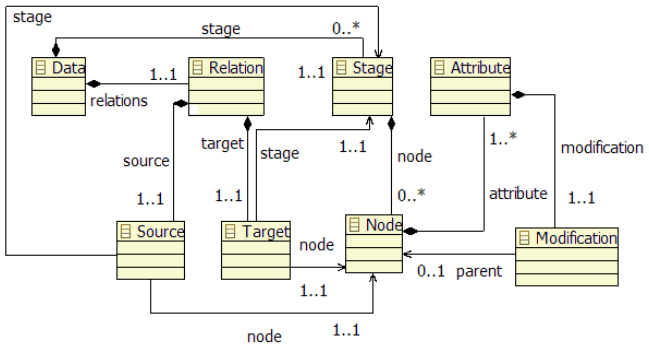

Figure 8: TraceVis Metamodel

```
    stage <- thisModule.MarkstageL
   ),
   targ: TRACEVIS!Target(
    node <- s.right.getReferredElement(),
    stage <- thisModule.MarkstageR
   )
 }
...
```

Target `Relation` elements are generated by the rules `NodeChangedClassDiff` and `RelationLink`. In particular, the former generates the link between the source difference model and the evolving metamodel. The latter generates the link between the evolving metamodel and the affected artefacts.

It is important to note that our approach is generic and can be used for supporting the *Change Impact Visualization* of any affected artefacts involved in the ecosystem. In fact, by observing the structure of `TraceVisGen` it is possible to identify a recurring pattern (shown in Listing 3) for each metaclass *MC* of the artefact metamodel *MM*. Thus, it is possible to generate the `TraceVisGen` transformation for each artefact using a higher-order transformation (HOT[3]) which takes the artifacts metamodel and generates the appropriate `TraceVisGen` transformation.

Listing 3: Recurring pattern in the artifact element node

```
rule NodeModule{
 from
  s : MM!MC
 to
  node : TRACEVIS!Node(...),
         ...
}
```

4.3 Generation of the TraceVis XML specifications

According to Figure 7, the `TraceVisModel` generated by means of the ATL transformation presented in the previous section is the input of the `TraceVisMM2XML` model-to-text transformation. Such a transformation is implemented by means of the Acceleo[4] code generator. Acceleo is a template-based approach for generating text from models. Acceleo templates identify repetitive and static parts of text, and embody specific queries on the source models to fill the dynamic parts.

Listing 4: TraceVisMM2XML transformation

```
...
[module generate('http://www.di.unavaq.org/Tracevis')
    /]
[template public generate(aData : Data)]
 [comment @main /]
 [file ('export.xml', false, 'UTF-8')]
 <?xml version="1.0" encoding="ISO-8859-1"?>
 <data>
  <stages>
   [for (s:Stage | aData.stages)]
    <stage id="[s.id/]" name="[s.name/]">
     [for (n:Node | s.node)]
     <node id="[n.id/]" inserttime="[n.inserttime/]">
      ...
     </node>
     [/for]
    </stage>
   [/for]
  </stages>
  <relations>
   [for (r : Relation | aData.relations)]
   <relation inserttime="[r.inserttime/]" type="
       Impact_On">
    ...
   </relation>
   [/for]
  </relations>
 </data>
 [/file]
[/template]
```

This Acceleo template in Listing 4 generates an XML file called *export.xml*. In particular, (i) for each stage in the source `TraceVisModel` it generates a `<stage>` tag, (ii) for each source instance of the metaclass `Node` it creates a `<node>` tag, and (iii) for each source instance of the metaclass `Relation` a target `<relation>` tag is generated.

5. DISCUSSION

By using the change impact visualization presented above, different recurrent patterns have been identified. In particular, the required changes in the affected artifacts are not always directly proportional to the changes in the metamodel as discussed in the following.

Uniform impact The example in Figure 9 is a small set of evolution steps where the delta model contains few connections with the evolving metamodel and there is a direct correspondence with the impact between metamodel and artifact. This situation is recurrent when a metamodel evolves by means of few atomic changes. In this case the mapping denotes one change in the metamodel correspondes to one change in the artifact, that is not strictly a 1-1 relation but the number of connections is similar in the two stages. This situation is strongly related to the artifact nature: e.g., changing a metaclass in the metamodel corresponds to a corruption point in a transformation, in particular in the rule where the metaclass in matched as input or output. On the contrary, considering a model as artifact the correspondence is different because likely the instances of the metaclass will be frequently instantiated in the model (see Figure 11).

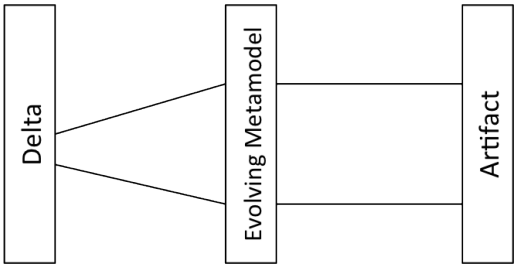

Figure 9: Uniform impact

Heavy metamodel evolution - Light impact on the artifact The situation shown in Figure 10 contains a lot of changes in the metamodel and less impact on the artifact. This situation is not very frequent but also not rare, according to our experience. This case implies that the involved elements in the metamodel are not referred in the artifact. For instance, in case of an extract superclass change[5] in the

[3]A HOT transformation is a special kind of transformation which has other transformations as input and/or output

[4]http://www.eclipse.org/acceleo/

[5]`http://www.metamodelrefactoring.org/`

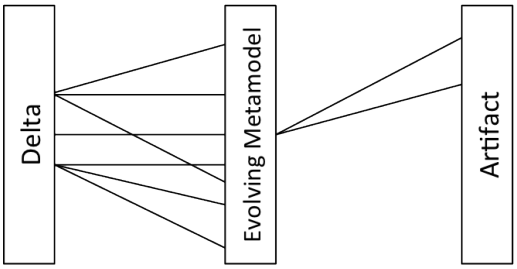

Figure 10: Heavy metamodel evolution - Light impact on the artifact

metamodel, all the attributes in the subclasses have to be removed, and a new one in the added super class has to be added. If the existing artefact is a model transformation, then we will have a light impact because the constructs relating to the extracted attribute will not be corrupted and still matched. Same situation in case of models where the the class instances continue to be valid because of the inheritance relation.

Light metamodel evolution - Heavy impact on the artifact The example in Figure 11 depicts a strong impact on the metamodel coming from a light metamodel evolution. A typical application of this category can be found in changing or renaming an attribute in a superclass. In case of ATL

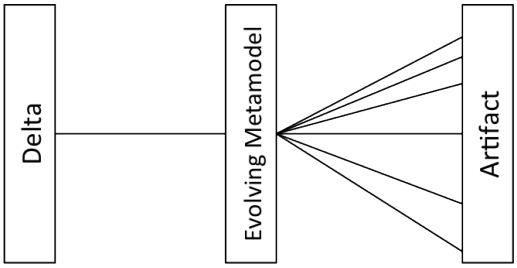

Figure 11: Light metamodel evolution - Heavy impact on the artifact

transformations, the renamed attribute can be duplicated in all the rules involving the subclasses and this leads to a strong impact for the duplicated information that has to be fixed. Removing an attribute from a metaclass can fall in the same category when this attribute is used many times in the artifact (e.g., in the case of helpers and rules in ATL transformations).

Heavy metamodel evolution - Heavy impact on the artifact The example in Figure 12 represents a very heavy impact on the metamodel coming from a heavy metamodel evolution. Such a situation is not common in literature, and represents that the metamodel has been heavily changed and consequently the existing artifact has been strongly corrupted. This example would be diffused if the metamodel evolution activity was not a step by step evolution.

For example this case would be produced if OMG were passing from UML 1.0 directly to UML 2.0 avoiding the intermediate versions. Since the metamodel definition is a cyclical activity where the metamodeler starts by defining concepts and refining the corresponding metamodel definition, this case is related to a well defined metamodel that undergoes strong changes coming from a turn upside down

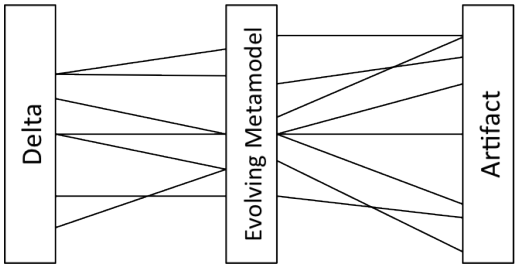

Figure 12: Heavy metamodel evolution - Heavy impact on the artifact

of the concepts in the application domain.

Non breaking changes with no impact The example in Figure 13 represents a set of non breaking metamodel changes that do not impact the existing artifact. We can imagine to transform an abstract metaclass into a real one. In this case, existing transformation rules are not affected. Of course this case can be considered as the best case that

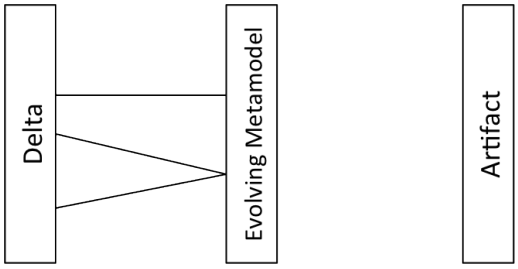

Figure 13: Metamodel non-breaking changes and no impact on the artifact

a metamodeler can see during the evolution phase, however this is a very rare case.

6. RELATED WORKS

Much research has been conducted on traceability, typically focused on software engineering. However the vast majority of the research is focused on either trace retrieval [18] or trace management [19] and usually neglects the visualization part. Trace retrieval is an activity that uses data mining techniques to generate the relations between artifacts created during the software engineering process. By analyzing keywords, relations can be defined between different parts of the artifact. Trace management focuses on maintaining traces during the software development process. In general, this is done by the authors or the maintainers of the artifacts. Results of trace retrieval or trace management are typically visualized using common visualization methods. According to Wieringa [20], visualization approaches can be categorized in matrices, cross-references and graph-based representations. Beyond these three types of visualization, there are only a few approaches to visualize traces in a different way.

The commercial application *TBreq* is focused on software engineering. The artifacts are listed horizontally and if a relation is defined between items of two artifacts an edge is

drawn. TBreq does not use the hierarchical structure of artifacts to reduce the size of the model visualization nor does it use any edge aggregation techniques or edge bundling to reduce cluttering. Although this works for small examples, large examples quickly give visual clutter. Moreover, due to the fact that individual items of an artifact are displayed as a list, only a limited number of items per artifact can be displayed. Nevertheless *TBreq* does convey traceability information since it gives a (partial) overview and can be used to quickly follow traces within the data. This tool can also show relations between two non-consecutive artifacts.

Pilgrim et. al. [21] use a three dimensional approach to gain insight in transformation chains containing different UML diagrams. Artifacts are all UML diagrams and are projected on a plane. Relations are visualized using edges between consecutive planes. Using a three-dimensional approach has little benefits over a two dimensional approach and still gives visual clutter when models become too large and too many relations are defined between planes. Moreover, using the third dimension will lead to occlusion problems when projecting the image to screen.

Marcus et al. [22] propose a visualization method that only shows information considered to be important to developers working on a part of a software project. The visualization method is similar to cross reference based methods and shows the various parts of an artefact as a rectangle, and related parts are assigned a similar colour. It is used to show relations of multiple artefacts, however it can only visualize a small number of dependencies and therefore cannot provide a global overview. Moreover, traces can only recursively be followed.

Briand et al. in [23] propose an approach to analyze the impact of the changes on the design model before applying the changes to the implementation model. The authors provided a classification of change types for three UML diagrams: class, sequence diagram and state machine diagrams. For each change type, an impact analysis rule is specified, describing how to extract the list of elements that are impacted by that particular change type. The definitions of change types and impact analysis rules are expressed in the Object Constraint Language (OCL). The propagation of changes to indirectly related entities is controlled by a distance measure, which is used to either stop the change propagation or to prioritize impact paths according to their depth. A prototype tool (iACMTool) was developed to automate and evaluate the feasibility of such an approach.

Fourneret and Bouquet [24] present an approach to perform model-based impact analysis for UML Statechart diagrams using dependence algorithms. The rationale of this type of analysis is to consider dependencies between transitions instead of states. In such context, two transitions are said to be data dependent if one transition defines a value of a variable that can be potentially used by the other transition and they are said to be control dependent if one transition may affect the traversal of the other transition. Based on this concept, the authors identified the data and control dependencies for UML statecharts and formulated the corresponding algorithms to be used in computing the dependence graphs of the statecharts elements. Accordingly, they identified and classified the possible changes to UML statecharts such as adding new transitions, deleting, and modifying existing transitions. It is worth to note that adding or deleting a transition can impact both the data and control dependence graphs while modifying a transition can impact only the data dependence graph. By having the two versions of statecharts (the original and the modified one) and their computed dependence graphs, the authors illustrated how their proposed dependence algorithms are used to extract the impacted elements. Our approach is different starting from the fact that is artifact-independent; in the cited works all those approaches are related to specific artifacts or particular set of models. The editor is a two dimensions editor that offers an advanced set of features like zooming and selection. Moreover the important characteristic is that for the first time we apply those features to the impact visualization in MDE ecosystems. Indeed given the independence characteristic of the tool chain, it is possible to replace the final transformation in order to provide the compatibility with other visualization tools.

This work is in part related also to coupled evolution process and the techniques and methodology for coupled evolution have been previously investigated in [25]. Coupled evolution is related to adapt models in response to metamodel changes [4, 26, 7, 6] and only recently, the problem of metamodel evolution/transformation co-adaptation has started to be investigated. Few attempts have been provided to deal with it in a dedicated way [8, 27, 28]. The methodology proposed in [5] has been extended and in part completed in this work adding the visualization activity as a new feature available in the methodology.

Evolution of modelling languages is also treated in [29], where the authors provide a taxonomy for evolution and discuss the various evolution cases for different kinds of modeling artifacts. The methodology discussed in this paper proposes to decompose the migration actions into primitive scenarios that can be resolved by the framework.

7. CONCLUSIONS

The problem of adapting the diversity of artifacts composing a metamodeling ecosystem when the metamodel undergoes modifications is intrinsically difficult. Trying to repair models, transformations, and editors without the guidance of a methodology can introduce inconsistencies and lead the ecosystem into an ungovernable status. This paper presented a generic approach to change impact visualization intended as a technique for visualizing the dependencies between Ecore metamodels and related artifacts in terms of traceability information. The modifications in a metamodel sometimes are not sustainable and being able to assess this aspect in advance can spare the modeler with unmanageable difficulties because of the intricacy of the adaptations. Compared with the work in [5] where a cost function is proposed, the presented technique conveys to the modeler a more intuitive notion of *intensity* and *locality* (see Sect. 5): the visualization permits to informally estimate how many adaptations are required in a restricted portion of an artifacts and especially the multiplicity of the impact on an artifact of a given metamodel change. Another contribution of the paper is the formalization of the TraceVis metamodel which permits to bridge Ecore models with this specific visualization tool.

In the near future, we are interested in the *live* monitoring of the change impact, i.e., as soon as a change is made on the metamodel information about the impact on related

artifacts in the ecosystem are automatically displayed in a context view. Despite the simplicity of the idea, its realization requires an operation recorder for registering the model differences in the metamodel, the incremental generation of the dependencies between the elements of the metamodel and of the related artifacts, and finally a proper presentation of impact analysis. The visualization can be enriched with the categorization of the refactorings in order to provide a more complete idea of the impact on the artifact. Representing this categorization with different colors can be useful to distinguish different forms of impact.

8. REFERENCES

[1] Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer **39**(2) (2006) 25–31

[2] Iovino, L., Pierantonio, A., Malavolta, I.: On the impact significance of metamodel evolution in mde. Journal of Object Technology **11**(3) (October 2012) 3:1–33

[3] Di Ruscio, D., Iovino, L., Pierantonio, A.: Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: ICGT. Volume 7562 of Lecture Notes in Computer Science., Springer (2012) 20–37

[4] Wachsmuth, G.: Metamodel Adaptation and Model Co-adaptation. In Ernst, E., ed.: Proceedings of the 21st ECOOP. Volume 4069 of LNCS., Springer-Verlag (July 2007)

[5] Di Ruscio, D., Iovino, L., Pierantonio, A.: A methodological approach for the coupled evolution of metamodels and ATL transformation. In: Proc. 6th International Conference on Model Transformation (ICMT'13). (2013)

[6] Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE - Automating Coupled Evolution of Metamodels and Models. (2009) 52–76

[7] Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: Automating Co-evolution in Model-Driven Engineering. In: 12th IEEE International EDOC Conference (EDOC 2008), Munich, Germany, IEEE Computer Society (2008) 222–231

[8] Levendovszky, T., Balasubramanian, D., Narayanan, A., Karsai, G.: A Novel Approach to Semi-Automated Evolution of DSML Model Transformation. In: Second International Conference on Software Language Engineering, SLE 2009, LNCS. Volume 5969., Denver, CO, Springer, Springer (05/2010 2010)

[9] Di Ruscio, D., Laemmel, R., Pierantonio, A.: Automated co-evolution of GMF editor models. In Malloy, B., Staab, S., van den Brand, M., eds.: 3rd International Conference on Software Language Engineering (SLE 2010). Number 6563 in LNCS, Springer, Heidelberg (October 2010) 143–162

[10] Di Ruscio, D., Iovino, L., Pierantonio, A.: What is needed for managing co-evolution in MDE? In: Proceedings of the 2nd International Workshop on Model Comparison in Practice, ACM (2011) 30–38

[11] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: Atl: A model transformation tool. Science of Computer Programming **72**(1-2) (2008) 31–39

[12] Cicchetti, A., Di Ruscio, D., Pierantonio, A.: A Metamodel Independent Approach to Difference Representation. Journal of Object Technology **6**(9) (October 2007) 165–185

[13] Di Ruscio, D., Iovino, L., Pierantonio, A.: Evolutionary togetherness: how to manage coupled evolution in metamodeling ecosystems. In: Intl. Conf. on Graph Transformations (ICGT 2012). Volume 7562 of LNCS., Springer (2012)

[14] van Ravensteijn, W.: Visual traceability across dynamic ordered hierarchies. Master's thesis, Eindhoven Univ. of Technology, The Netherlands (2011)

[15] Amstel, M., Brand, M., Serebrenik, A.: Traceability visualization in model transformations with tracevis. In Hu, Z., Lara, J., eds.: Theory and Practice of Model Transformations. Volume 7307 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 152–159

[16] Brun, C., Pierantonio, A.: Model Differences in the Eclipse Modeling Framework. UPGRADE, The European Journal for the Informatics Professional **9**(2) (April 2008)

[17] Pierantonio, A., Iovino, L., Di Rocco, J.: Bridging state-based differencing and co-evolution. In: Models and Evolution Workshop - 15th International Conference on Model Driven Engineering Languages and Systems. (September 2012)

[18] Lucia, A.D., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. **16**(4) (September 2007)

[19] Lozano, A., Pinter, R.Y., Rokhlenko, O., Valiente, G., Ziv-ukelson, M.: Seeded tree alignment and planar tanglegram layout

[20] Wieringa, R.: An introduction to requirements traceability (September 1995) Acknowledgements: This report benefited from input given by Eric Dubois, Anthony Finkelstein and Hanna Luden.

[21] Pilgrim, J., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and visualizing transformation chains. In Schieferdecker, I., Hartman, A., eds.: Model Driven Architecture âĂŞ Foundations and Applications. Volume 5095 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 17–32

[22] Marcus, A., Xie, X., Poshyvanyk, D.: When and how to visualize traceability links? In: Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering. TEFSE '05, New York, NY, USA, ACM (2005) 56–61

[23] Briand, L.C., Labiche, Y., O'Sullivan, L., Sówka, M.M.: Automated impact analysis of uml models. J. Syst. Softw. **79**(3) (March 2006) 339–352

[24] Fourneret, E., Bouquet, F.: Impact analysis for uml/ocl statechart diagrams based on dependence

algorithms for evolving critical software. Laboratoire dâĂŹInformatique de Franche-Comté, Besançon, France, Tech. Rep. RT2010-06 (2010)
[25] Favre, J.M.: Meta-Model and Model Co-evolution within the 3D Software Space. In: Procs. of the Int. Workshop ELISA at ICSM. (September 2003)
[26] Rose, L.M., Kolovos, D.S., Paige, R.F., Polack, F.A.C.: Model Migration with Epsilon Flock. In: ICMT. (2010) 184–198
[27] García, J., Díaz, O., Azanza, M.: Model transformation co-evolution: A semi-automatic approach. In: SLE. (2012) 144–163
[28] Méndez, D., Etien, A., Muller, A., Casallas, R.: Transformation migration after metamodel evolution. In: International Workshop on Models and Evolution (Me'10) - ACM/IEEE MODELS'2010. (2010)
[29] Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Science of Computer Programming **76**(12) (2011) 1223 – 1246