Uncertainty management with extra-functional qualities in multi-artefact co-evolution

Francesco Basciani[†], **Davide Di Ruscio**[†], **Ludovico Iovino**[*], **and Alfonso Pierantonio**[†]
[†]Università degli Studi dell'Aquila, Italy
[*]Gran Sasso Science Institute, Italy

ABSTRACT
Metamodels are at the core of any model-driven projects and give the experts a linguistic means to abstractly represent the problem domain's instances. Metamodels are prone to modifications due to new insights emerging from the domain, improvements, and features being added to the modeling language over time. Metamodel evolution may induce severe repercussions over the related artifacts, which might need to be consistently adapted. The co-evolution problem poses another issue related to the epistemic uncertainty arising as a response to a metamodel change where many different consistency restoration procedures are possible, e.g., restoring models and transformations. In this work we rely on the notion of *information loss* (IL) to understand which migration combinations are preferable to others in coupled evolution, by offering a ranking of the possible solutions. The IL denotes that part of the knowledge contained in the source models cannot be conveyed or translated into the target models, using a selected transformation, resulting in a loss of information. Such aspect represents an extra-functional quality that can be used for ranking the selection of certain transformation migrations in the context of multiple available alternatives and in combination with model migrations. Information loss induced by a (migrated) transformation can be defined as the amount of information lost executing the (migrated) transformation compared to the original output model obtained by executing the original transformation. The proposed approach is supported by a prototype tool that we demonstrate and validate.

KEYWORDS MDE , Parallel Coupled-evolution , Information loss

1. Introduction

Model-Driven Engineering (MDE) helps tame the sheer complexity of modern software systems by leveraging abstraction and automation (Schmidt 2006). MDE is becoming popular not only in academia but also in industry (Hutchinson et al. 2014) as testified by a broad range of companies that employed MDE in different application domains including automotive, telecommunications, defense, aerospace, Industry 4.0, and healthcare (Vallecillo 2014). Regardless whether abstract descriptions of complex functionality, complementary views (e.g., behavioral versus structural), or vertical refinement of high-level requirements models into design models and eventually down to executable code, model-driven techniques offer sound and technically consistent solutions. Metamodels are at the core of any model-driven projects and give the experts a linguistic means to abstractly represent the problem domain's instances. The domain metamodels allow modelers to define a complete modeling environment that comprehends not only models, but also editors (and the related syntax), model-to-model transformations, analyzers, code generators, and more (Bézivin 2005). Like any long-living software artifact, metamodels are prone to modifications due to new insights emerging from the domain, improvements, bug-fixing, and features being added to the modeling language over time. However, changing a metamodel may induce severe repercussions over the related

JOT reference format:
Francesco Basciani, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. *Uncertainty management with extra-functional qualities in multi-artefact co-evolution.* Journal of Object Technology. Vol. 20, No. 3, 2021. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2021.20.3.a2

artifacts whose validity restoration requires specialized support. As a consequence, a metamodel lock-in may quickly emerge where modelers must face the inability to keep the overall environment consistent (Iovino et al. 2012). The problems induced by metamodel evolutions considerd in this paper have affected also well-known languages including UML. Despite its status as an industry standard, the UML metamodel has been subject of several evolutions to increase the expressiveness of the language, and to accommodate unforeseen requirements coming from different kinds of stakeholders including industrial practitioners (Thompson & Platt 2015).

Over the last years, a corpus of literature studies has accrued that addresses the problem of co-evolution. Most of these studies are devoted to the co-evolution of metamodels and models (see (Hebig et al. 2016) for a survey). Other approaches are dealing with the co-evolution of transformations (Garcés et al. 2013; Kessentini, Sahraoui, & Wimmer 2018), editors (Falzone & Bernaschina 2018), and even syntaxes (Di Ruscio et al. 2013). An interesting aspect that has been previously investigated in (Di Ruscio et al. 2016; Di Ruscio et al. 2017) is related to the epistemic uncertainty arising as a response to a metamodel change where many different consistency restoration procedures are possible. A typical example is when the multiplicity of an association in a metamodel is restricted: it gives place to multiple ways of selecting the exceeding associations to be removed from the instance models (Kessentini, Wimmer, & Sahraoui 2018). Analogously, the migration of transformations can be conducive to bound uncertainty too. For instance, if we merge two attributes of a metaclass, and we have two bindings in transformation rules predicating over those attributes, which one do we need to consider in the migration? Moreover, combining all these alternatives of migrated artifacts can lead to a matrix of possible combinations. The modeler then has to carefully inspect and detect the *best* migration and artifacts with respect to her final intent.

This paper proposes an approach to migrate both models and model-to-model transformations in response to a metamodel revision. Since each different artefact can give place to multiple migration policies, the method introduces a ranking technique to reduce the combinations to explore, as one of the possible criteria in applying parallel-coupled evolution of models and transformations. The migration ranking relies on a notion of *information loss* (Basciani et al. 2018) (IL), a measure that might suggest a criterion for selecting migration combinations over others. The IL represents an extra-functional quality denoting the amount of knowledge in a source model that cannot be conveyed to the corresponding target model resulting in a loss of information. Therefore, it provides the modeler with a ranking for selecting specific transformation migrations in the context of multiple available alternatives and in combination with model migration alternatives. The information loss induced by a (migrated) transformation can be defined as the amount of information lost executing the (migrated) transformation compared to the output model obtained by executing the original transformation. The approach permits combining, visualizing, and representing the possible combination of migration alternatives for corrupted artifacts using weaving models. Each migration is assigned the corresponding information loss computed according to the technique in (Di Ruscio et al. 2016; Di Ruscio et al. 2017). The approach is generic and can be extended to other extra-functional qualities and different kinds of artefacts, including editors and code generators. It is important to remark that the proposed approach can manage the evolution of modeling artifacts, which do not include OCL constraints in their definition. Considering them in the proposed method is planned as future work.

Structure of the paper The paper is organised as follows: Section 2 shows that the evolution of the metamodel might affect both models and transformations. Section 3 shows how multiple alternatives of migration are possible when considering models and transformations in the co-evolution phase. Section 4 demonstrates that this parallel activity might induce information loss if considered in combination of models and transformations; Section 5 proposes an approach able to generate, visualize and rank existing co-evolution alternatives based on information loss, while in section 6 we describe the tool supporting the presented approach. Section 7 presents the evaluation of the proposed techniques. After a summarizing discussion given in Section 8, related work is presented in Section 9. Section 10 draws the conclusions and future work.

2. Motivating scenario

As shown in Fig. 1, typical MDE settings consist of different kinds of artifacts including models, metamodels, transformations, and code generators. All these constituting elements are linked by specific relationships and all together give place to a metamodeling ecosystem, i.e., *"a metamodel-centered environment whose entities are traditionally subject to distinct evolutionary pressures but cannot have independent life-cycles"* (Di Ruscio et al. 2012).

Similarly to any software artifacts, metamodels can evolve over time to meet unforeseen requirements or to simply improve related quality factors (Bettini et al. 2019). When metamodels are changed, ripple effects might occur on related artifacts included in the ecosystem, which thus needs to be co-evolved to recover the lost consistence. For instance, the explanatory scenario shown in Fig. 1 contains the `Company` metamodel, which is shown in Fig. 2. It underpins the definition of several

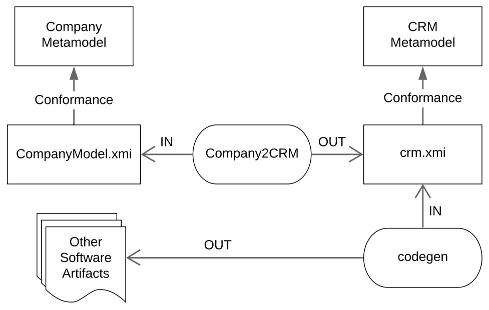


Figure 1 Explanatory metamodeling ecosystem

artifacts, including the model `CompanyModel.xmi`[1] shown in Fig. 4 and the transformation `Company2CRM` (reported in Listing 1) to generate models conforming to the `CRM` metamodel from source company specifications.

According to the metamodel given in Fig. 2, a company is made of `Persons` that can be distinguished as `employees` or `clients` based on their `position`. Each person can have an assigned `Project`, which can be in turn `European` or `National`. A project is classified with a `Category`, that can have multiple assigned `Topics`. A company is also characterized by the `Address` of its headquarter and can have multiple working `Divisions`, further classified in `Units` or `ServiceLines`.

Figure 3 shows the `CRM` metamodel, which permits to define CRM (Customer Service Management) applications that consist of `Accounts` of `Clients` or `Workers`, that can be `Grouped`. The CRM also keeps track of the currently available `Projects`.

As previously mentioned, the `Company2CRM` transformation is also part of the explanatory ecosystem and it consists of 7 transformation rules developed in ATL to generate `CRM` elements from source `Company` specifications (see Listing 1).

Listing 1 The explanatory `Company2CRM` ATL Transformation

```
module Company2CRM;
create OUT : CRM from IN : Company;
rule Company2CRM{
 from s: Company!Company
 to t: CRM!CRM(
  address <- 'www.'+s.name+'.com/'+s.address.city,
  accounts <- s.persons,
  projects<-s.projects,
  groups<- s.lines
 )
}
rule Person2Worker{
 from s: Company!Person(s.position=#employee)
 to
  t: CRM!Account(
   username <- s.firstname+'.'+s.lastname ,
   group<-s.employed
  ),
  t1: CRM!Worker(
   account<-t,
   name <- s.firstname+' '+s.lastname
  )
}
rule Person2Client{
 from s: Company!Person(s.position=#client)
 to
  t: CRM!Account(
   username <- s.firstname.toLower()+'.'+s.lastname.toLower(),
   group<-s.employed
  ),
  t1: CRM!Client(
   account<-t ,
   name <- s.firstname+' '+s.lastname
  )
}
rule European2Project{
 from s: Company!European
 to t: CRM!Project(
  name<-s.name,
  area<- s.related.name,
  budget<-s.budget
 )
}
rule National2Project{
 from s: Company!National
 to t: CRM!Project(
  name<-s.name,
  area<- s.related.name
 )
```

[1] The .xmi extension of the model refers to the XML-based representation of models conforming to metamodels defined in Ecore (EMF).

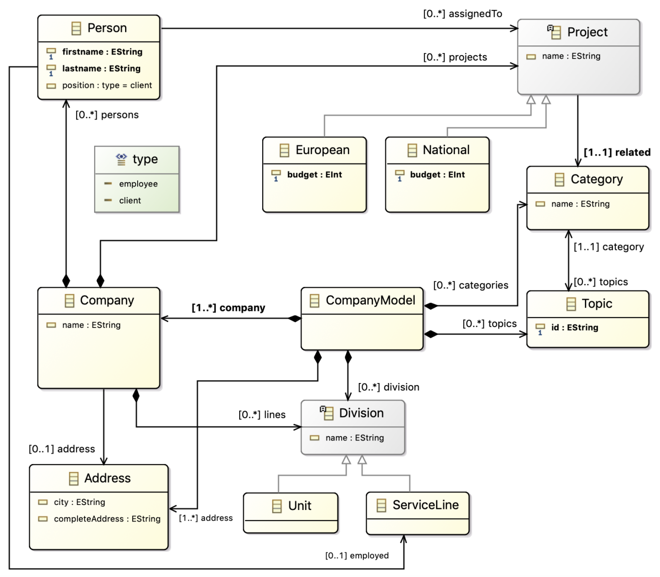


Figure 2 Company Metamodel

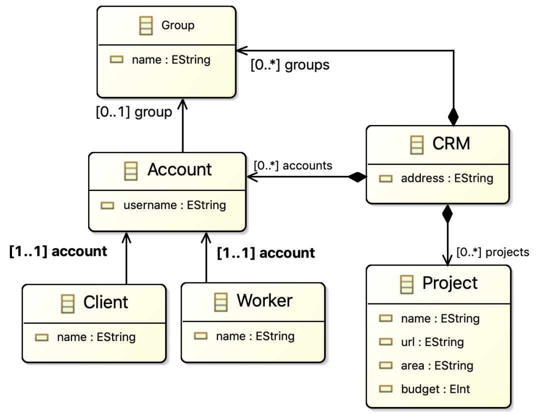


Figure 3 CRM metamodel

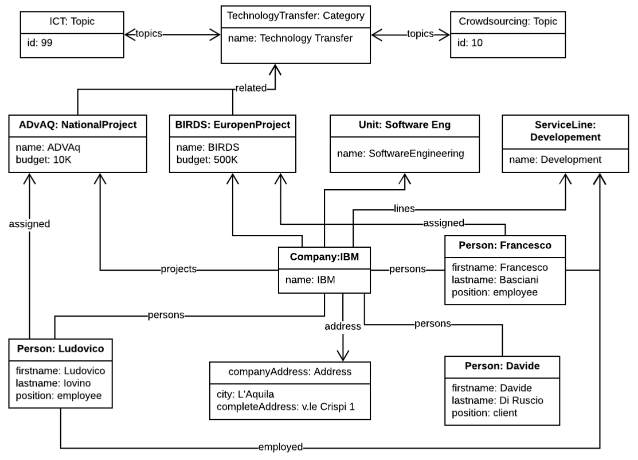


Figure 4 Company Model

```
50 }
51 rule Unit2Group{
52  from s: Company!Unit
53  to t: CRM!Group(
54   name<-s.name
55  )
56 }
57 rule ServiceLine2Group{
58  from s: Company!ServiceLine
59  to t: CRM!Group(
60   name<-s.name
61  )
62 }
```

Figure 5 shows an evolved version of the Company metamodel, which has been changed by applying a number of refactorings. The proposed refactorings have been applied for the purpose of highlighting evolution aspects, which are relevant for the sake of presentation of the proposed approach. Because of the performed changes, all the artifacts that are defined in terms of the metamodel Company might have been affected and in this case it has to be migrated. In particular, the refactorings that have been operated on the initial version of the metamodel Company are explained below together with the effects on the CompanyModel.xmi model (represented in Fig. 4) and on the Company2CRM transformation (reported in Listing 1).

R1: Enumeration To Subclasses. This refactoring converts an enumeration to new classes. In more details, by referring to Fig. 2: *i)* the class Person becomes abstract, *ii)* the new metaclasses Employee and Client are added, and *iii)* the attribute position with its enumeration is deleted as effect of introducing the subclasses. Due to the application of such refactoring the transformation Company2CRM gets corrupted. In particular, the lines 13 and 25 are responsible of the error *"Feature position does not exist in Person"* that is raised at run-time when executing the transformation with the new version of the metamodel. Also the CompanyModel needs to be migrated as the three different instances of the metaclass Person can no longer be instantiated, being the metaclass Person abstract in the new version of the metamodel (see Fig. 5).

R2: Rename Class. This evolution renames the class Company as Organisation. Due to this refactoring, another run-time error is raised by the transformation due to line 4. It refers to the metaclass Company, which is no longer existing. Also the model is affected concerning the Company instance, which should be retyped as Organisation.

R3: Collapse Hierarchy. Collapsing an existing hierarchy is a model change occurring when the subclasses do not add specific information with respect to the superclass. In the motivating example, the subclasses of the metaclass Project share a common attribute budget. The refactoring collapses the hierarchy, i.e., it makes the metaclass Project concrete and moves the attribute budget into it. The transformation rules responsible for translating European and National projects to target CRM Projects become invalid since the two classes used as matching elements — European and National in lines 37 and 45 in Listing 5, respectively — are no longer available in the Company model. Each instance in the model referring to the European and National metaclasses as types have to be retyped. In the shown example, this is the case of the *BIRDS* and *AdvAq* elements as shown in Fig. 4.

R4: Merge Attributes. Two or more attributes can be merged into a single one. For instance, the attributes firstname and lastname in the Person metaclass share the same type, and in the evolved version of the metamodel a new attribute fullname replaces them. Such refactoring has an impact on the Person2Worker and Person2Client transformation rules, where the attributes firstname and lastname are referred (lines 16, 21, 28 and 33). Moreover, also the Person instances are corrupted where the value for firstname and lastname are set.

R5: Inline Class. Often this refactoring is needed after the features of one metaclass are "transplanted" into other metaclasses, leaving that original metaclass no longer needed. For instance, by applying an inline class to the Company model and specifically on the reference address of the Company metaclass, the evolved version of the model should contain the structural features city and completeAddress directly in the Company metaclass, and the unused metaclass Address should be removed. Due to such refactoring, the transformation rule Company2CRM gets a corrupted binding in line 6 where the navigation to s.address is inconsistent with the evolved version of the Company metamodel. This is because the reference address is no longer navigable. Also the model presents inconsistencies and in particular shows the *IBM Company* instance having a reference to the metaclass Address that is no longer existing in the new version of the corresponding metamodel.

R6: Change Reference Type (first application). This change is applied when a specific reference is re-typed with another class. This refactoring corresponds to the attribute type change, e.g., String to Int, but at the reference level. For instance, the reference related of the class Project typed with the metaclass Category in the initial version of the metamodel, is then re-typed to refer the metaclass Topic. The affected transformation rule is listed in lines 40 and 48, where the OCL expression s.related.name is inconsistent, since the new referenced class no longer contains the name attribute. Also two references in the model referring the instance of the Category metaclass (i.e., the *Technology Transfer* element) are no longer valid.

R7: Change Reference Type (second application). In the running example, another application of change reference type occurs (referenced in the following as R7). In particular, the employed reference in the Person class of type ServiceLine has been retyped as Unit. Such refactoring affects the transformation rules Person2Worker and Person2Client, due to the expression listed in lines 17 and 29, i.e. s.employed. Moreover, the instances of the metaclass Person in the model that have a reference to instances of ServiceLine have to be adapted in order to re-establish the conformance.

3. A multitude of possible migrations

Due to the metamodel changes described in the previous section, the whole ecosystem needs to be migrated. In particular, both

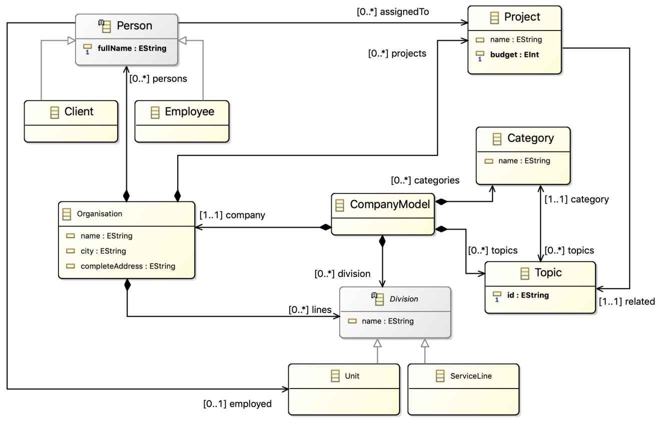


Figure 5 Evolved Company Metamodel

the model `Company.xmi` and the transformation `Company2CRM` need to be adapted so to be conformant with the new version of the `Company` metamodel. The adaptations of modeling artifacts that have to co-evolve with respect to the changes operated on the corresponding metamodels are not unique and more than one migration strategy can be possible. In (Di Ruscio et al. 2016) authors proposed an approach to identify and select migration alternatives to be applied when models affected by metamodel evolution have to be adapted. However, modeling artifacts do not exist alone and a satisfactory solution should consider all of them together in ecosystems. In our scenario, exemplified in Fig. 6, different model and transformation adaptations can be performed and depending on the selected migration strategies, the content of the target model `crm.xmi` will be produced accordingly.

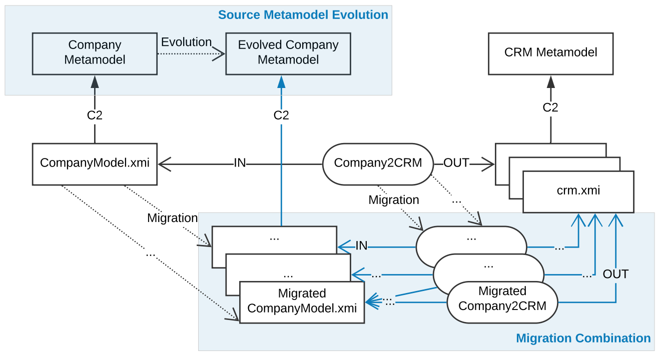


Figure 6 Explanatory scenario of different ecosystem adaptations

In the following we describe the possible migrations that can be operated on `Company.xmi` and `Company2CRM` in response to the refactoring of the `Company` metamodel. For the sake of readability, with $R_i m_j$ we refer to the migration alternative j for the model `Company.xmi` in response of the refactoring R_i operated on the `Company` metamodel. Similarly, with $R_i t_j$ we refer to the migration alternative j of the original `Company2CRM` transformation in response of the operated metamodel refactoring R_i. In the following, the alternative model and transformation migrations are presented for each operated metamodel refactoring.

R1: Enumeration To Subclasses Due to such metamodel refactoring, the `Company.xmi` model can be migrated in three possible ways: *i)* we can decide to migrate the existing instances of the `Person` metaclass to one of the introduced subclasses, i.e., `Employee` ($R_1 m_1$) or `Client` ($R_1 m_2$), or depending on the value of the attribute `position`, we can migrate the `Person` instances with `position = #employee` as `Employee` elements, and those with `position = #client` to `Client` instances ($R_1 m_3$).

Concerning the migrations of the `Company2CRM` transformation, it is possible to make it working with the new version of the `Company` metamodel by operating at least one of the following changes: ($R_1 t_1$) we convert the matched rule having as input pattern `Company!Person(s.position=`*#employee*`)` and the one with `Company!Person(s.position=`*#client*`)` to a rule having as input pattern `s: Company!Employee;` ($R_1 t_2$) we convert the matched rule having as input pattern `Company!Person(s.position=`*#employee*`)` and the one with `Company!Person(s.position=`*#client*`)` to a rule having as input pattern `s: Company!Client;` ($R_1 t_3$) we convert the matched rule having as input pattern `Company!Person(s.position=`*#employee*`)` to `s:Company!Employee` and the one having `Company!Person(s.position=`*#client*`)` to a rule having `s: Company!Client`. The combination of these model and transformation alternatives can give place to 9 different `crm.xmi` models in output.

R2: Rename Class The co-evolution alternatives for this refactoring offer less options than the previous one, being a simple renaming. The possible co-evolutions for the model can be as follows: retype all the instances of `Company` to `Organization` ($R_2 m_1$) or remove all of them from the model ($R_2 m_2$). Concerning the `Company2CRM` transformation it is possible to migrate it by changing the type of the input pattern of the affected rule to `Organisation` ($R_2 t_1$) or by dropping the existing rule from the transformation ($R_2 t_2$).

R3: Collapse Hierarchy In response of such metamodel refactoring, the `Company.xmi` model can be migrated by retyping all the `European` and `National` project instances as `Project` ones ($R_3 m_1$). Another possible migration can be keeping only the `European` instances and changing them as `Project` ones ($R_3 m_2$) or do the same for the `National` ones ($R_3 m_3$).

The `Company2CRM` transformation contains two rules having as input pattern the metaclasses `National` and `European`. Such rules are identical except for the bindings related to the `budget` of the project that in case of national ones is omitted. By applying the refactoring R3 we have two possible migration options: maintaining the `European2Project` rule and changing the input pattern type to `Project` ($R_3 t_1$) or maintaining the `National2Project` rule ($R_3 t_2$). Another option can be removing the affected rules ($R_3 t_3$).

R4: Merge Attributes Due to the merging of the attributes `firstname` and `lastname` in the `Person` metaclasses, possible migrations for the `Company.xmi` model can be as follows

: (R_4m_1) merging the values of the attributes `firstname` and `lastname` values to set the `fullName` attribute of all the existing `Person` instances; (R_4m_2) keeping only the value of the `firstname` attribute to set `fullName`; (R_4m_3) keeping only the value of the `lastname` attribute to set `fullName`;

Concerning the `Company2CRM` transformation, it can be migrated in different manners. In particular, since the merged attributes are part of the same expression in the transformation binding, i.e. `s.firstname+'.'+s.lastname` or `s.firstname+' '+s.lastname`, it is possible to perform the following alternative migrations: (R_4t_1) replacing this expression with the opposite one, e.g., `s.fullName.split(' ').get(0) +'.'+ s.fullName.split(' ').get(1)`; (R_4t_2) replacing the occurrences of the couple `firstname` and `lastname` with the first part of `fullName` (`s.fullName.split(' ').get(0)`); (R_4t_3) keeping only the expression related to `lastname` e.g., `s.fullName.split(' ').get(1)`.

R5: Inline Class In response to such metamodel refactoring, the `Company.xmi` model can be migrated in different manners including the following: (R_5m_1) the set of values in the instances of the inlined class, i.e. `Address`, are transferred to the values of the attributes moved to the source class, in this case `completeAddress` and `city`; (R_5m_2) the moved attributes will be empty in the new instances.

Concerning the `Company2CRM` transformation, (R_5t_1) the binding `'www.'+s.name+'.com/'+s.address.city` can be migrated to the corresponding expression `'www.'+s.name+'.com/'+s.city`, or (R_5t_2) the following expression can be considered `'www.'+s.name+'.com/'` by truncating the affected portion of the initial expression.

R6: Change Reference Type in Project To fix the `Company.xmi` model to make it conforming to the new version of the metamodel, it is possible to perform at least the following migrations: (R_6m_1) removing the corrupted references to the `Category` instances; (R_6m_2) selecting by means of some heuristic the topics assigned to `Category` instances that need to be kept; or (R_6m_3) keep the last `Category` instance that was created in the model.

The `Company2CRM` transformation can be migrated in different manners including the following ones: (R_6t_1) the `area` $\leftarrow$ `s.related.name` binding relates to the `name` attribute, which is not defined in the class `Topic`, so we can propose to replace the expression with the first attribute of the same type, i.e., `area` $\leftarrow$ `s.related.id`; (R_6t_2) another option is navigating with the expression `area` $\leftarrow$ `s.related.category.name` to obtain the same semantic of the original transformation; (R_6t_3) another possible adaptation consists of removing the binding containing the `related` attribute.

R7: Change Reference Type in Person Such a metamodel refactoring compromises only the conformance of the `Company.xmi` model and does not have any ripple effects on the `Company2CRM` transformation. The `Company.xmi` model can by migrated as follows: existing instances of `Person` having the reference `employed` set to an instance of `ServiceLine`

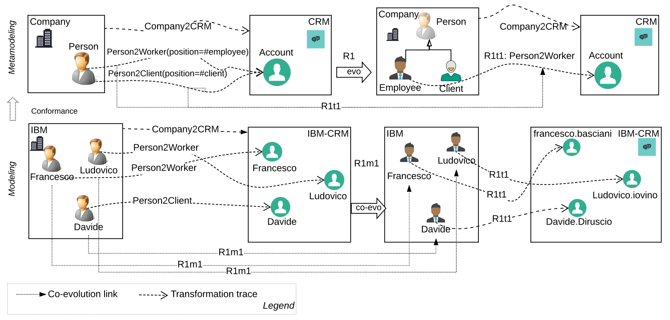


Figure 7 (R_1m_1,R_1t_1) migration pair with no induced information loss

are redirected to instances of `Unit` randomly (R_7m_1) or just removed (R_7m_2).

Selecting the migrations to be operated on both models and transformations in response to the refactorings performed on the corresponding metamodels is not an easy task and as previously described, the resulting ecosystems can be different depending on the given choices. In the next section, we present a possible criteria that can be considered for supporting such a daunting task.

4. Use of information loss to characterize ecosystem migrations

In this section we present *information loss (IL)* as a possible criteria, which can be considered when selecting the model and transformation co-evolutions that have to be operated to restore the consistency of the corrupted ecosystem.

In (Basciani et al. 2018), information loss has been considered as a quality aspect used to characterize alternative model transformation chains bridging the same source and target metamodels. Intuitively, in that work, the information loss has been defined as a value, which has *"to increase (possibly linearly) as the number of instances (along with the owned features) conforming to metaclasses of the source metamodel that are not covered by the transformation chain increase"*. Similarly, in this paper we want to propose information loss as a measure to characterize alternative ecosystem migrations, and it's value represents the (target) model instances, which are lost by the selected pairs of (source) model and transformation migrations.

By referring to the motivating scenario and to the metamodel refactoring R1 presented in Section 2, in the following we describe how it is possible to characterize pairs of corresponding model and transformation migrations with respect to the induced information loss. In particular, different explanatory migrations are described with respect to the induced information loss, which can be total, partial or it can be that no information loss occurs.

Figure 7 graphically shows the application of the (R_1m_1,R_1t_1) migration pair. Concerning the original transformation (see Listing 1), the operated metamodel refactoring affects the `Company2CRM` rule, which generates `CRM` elements out of input `Company` instances. To this end, two additional rules are also involved, i.e., `Person2Worker` (line 12) and `Person2Client` (line 24). The former creates an `Account` element for each input `Person` of the company having the attribute `position` set to `employee` (similarly for clients), and this is shown by the transformation trace links (also named mappings hereafter and shown as dotted arrows in the `Metamodeling` layer of Fig. 7) between the input and target metamodels.

The lower side of Fig. 7 shows some elements of the original `Company.xmi` model, i.e., the `Company` named *IBM*, two hired `Persons` (i.e., *Ludovico* and *Francesco* as `Employee` instances) and the `Client` named *Davide*. By executing the original transformation, on the initial version of the source model, three `Account` elements are generated by enabling the access to the CRM, one for each person registered in the company, where they can put their own personal information. Due to the metamodel refactoring R1, the model gets corrupted since the instances of `Person` can no longer exist in the model (being declared abstract in the evolved metamodel as shown in Fig. 4 and the instances declared in Fig. 7 in the modeling layer). Moreover, the rule `Person2Worker` and `Person2Client` should be adapted in order to remove the `position` used in the input pattern condition that can no longer be navigated on `Person` elements (see lines 12 and 24 of Listing 1). For this reason we can propose some different migration alternatives for the affected model and transformation. In particular, concerning the original `Person2Worker` rule the first proposed alternative R_1t_1 is shown in Listing 2, where from two matched rules with corrupted input patterns, we create a single rule with input pattern matching `Employee` elements.

```
rule Person2Worker{
 from s: Company!Employee
 to t: CRM!Account(
 username <- s.firstname+'.'+s.lastname,
 group<-s.employed
 ),
...
}
```

Listing 2 Alternative R_1t_1

As can be noticed in Fig. 7, where the complete picture of the transformation is shown, also the model can be migrated in different ways, and in this case, all the `Person` instances have been migrated to `Employee` ones. If we execute the migrated transformation R_1t_1 with the chosen migrated model R_1m_1, we obtain a target model with no information loss, in fact all the original elements in the source model have been considered and preserved in the target one (thus, no information loss occurs).

Figure 8 shows another migration pair consisting of the previously considered R_1m_1 model migration, together with R_1t_2. In such transformation evolution, the original rules having `Person` as input source pattern has been changed to match `Client` instances, to make the transformation conforming again to the `Company` metamodel, as shown in Listing 3. In this specific case the combination of R_1m_1 and R_1t_2 induces a total information loss, since the instances of `Employee` are not matched by the evolved transformation rule, and consequently the produced output model is empty (thus, a total information loss occurs).

```
 rule Person2Worker{
  from s: Company!Client
  to t: CRM!Account(
```

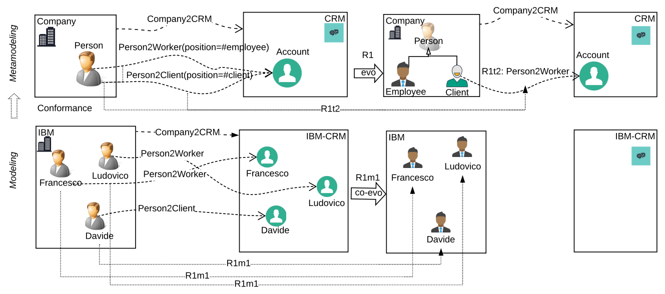


Figure 8 (R_1m_1,R_1t_2) migration pair with induced total information loss

```
  username <- s.firstname+'.'+s.lastname ,
  group<-s.employed
  ),
  ...
 }
```

Listing 3 Alternative R_1t_2

Figure 9 shows another migration pair consisting of the model adaptation R_1m_3 and of the transformation evolution R_1t_1. Due to R_1m_3, some instances of the original model are migrated to `Employees` and others to `Clients`, depending on the value of the attribute `position`. By combining such model evolution with the previously described R_1t_1 transformation, the situation that can be obtained in the corresponding generated model is shown in Fig. 9, which is characterized by a partial information loss, meaning that we loose part of the information originally contained in the input model. In particular, by comparing the original model obtained by executing the transformation with the one obtained here, we receive two instances instead of three, namely *ludovico.iovino* and *francesco.basciani*, loosing the instance translated from the client *Davide*.

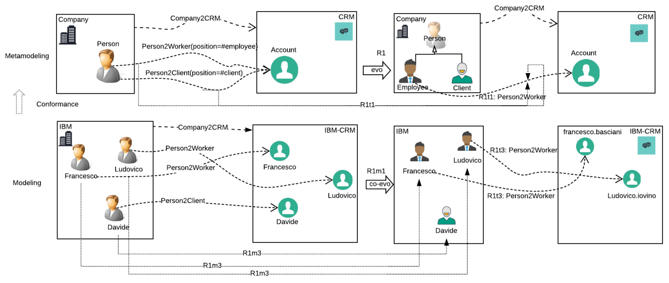


Figure 9 (R_1m_3,R_1t_1) migration pair with induced partial information loss

Figure 10 shows another migration case, which does not induce any information loss. In particular, the migration pair consisting of R_1m_3 and R_1t_3 shown in Listing 4 is applied. The migrated source model consists of two employees and of one client, which will be translated into accounts using the rules of the transformation R_1t_3, with a corresponding no information loss as result.

```
rule Employee2Worker{
 from s: Company!Employee
 to t: CRM!Account(
 username <- s.firstname+'.'+s.lastname ,
```

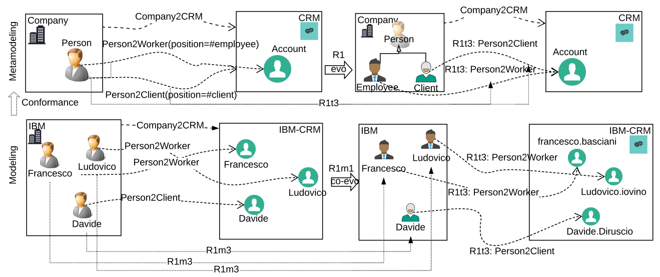


Figure 10 (R_1m_3,R_1t_3) migration pair with no induced information loss

```
group<-s.employed
),
...
}
rule Client2Worker{
from s: Company!Client
to t: CRM!Account(
username <- s.firstname+'.'+s.lastname ,
group<-s.employed
),
...
}
```

Listing 4 Alternative R_1t_3

The cases previously discussed permit to grasp the message that depending on the selection and combination of alternative migrations, it is possible to obtain different results, which can be characterized by an induced information loss. In the next section we propose an approach, which permits to rank pairs of model and transformation migrations according to the loss of information that they induce.

5. Ranking co-evolution strategies

In this section we propose an approach to automatically analyze pairs of model and transformation alternatives with the aim of supporting modelers that have the responsibility of adapting ecosystems, which have been affected by metamodel refactorings. Modelers are thus provided with a tool supported method, which can rank migration pairs with respect to the induced information loss. The proposed technique underpins and extends the solutions presented in (Di Ruscio et al. 2016; Di Ruscio et al. 2017). They are able to manage alternatives in model and transformation migrations, by means of dedicated weaving models (Bézivin et al. 2005). In more details, the approach presented in (Di Ruscio et al. 2016) allows the modeler to represent all the available alternatives for model migrations in a single compact model. Then the resulting solution is transformed into a feature-based representation that offers graphical features to select the best solution and generate the desired migration strategy for models. The same can be done for model transformations by using a variability model presented in (Di Ruscio et al. 2017). These two approaches deal with single-artifact adaptation and does not consider the effect of applying pairs of migration strategies.

Figure 11 depicts an overview of the proposed approach, which consists of different steps that modelers are supposed to perform as described in the following. In particular, `Specification of Variability Model` is the first step of the process and it is based on a similar activity described in (Di Ruscio et al. 2017) for expressing the migration alternatives for transformations, and in (Di Ruscio et al. 2016) for representing alternatives of model migrations. Such variability models are labeled with $\boxed{\text{VWM}}$ (for the input model m) and $\boxed{\text{TVWM}}$ (for the input transformation T). These two variability models can be combined to generate and specify a combined migration of an input model $\boxed{\text{m}}$ and a transformation $\boxed{\text{T}}$.

The generated weaving model $\boxed{\text{Comb VWM}}$ allows the modeler to express all the possible combinations of migration alternatives together with the induced information erosion as discussed in the previous section. In particular, the $\boxed{\text{Comb VWM}}$ weaving model is used as input for ranking all the possible combined migrations and in case filter out those that are below a threshold given by the user (see the `Migration configuration` phase). The output of such an activity is the selected pair of model and transformation migrations, which are used to generate the actual migration programs (see the phase `Generation of the migration program`) that get executed during the last step of the process referred as `Execution of the migration` in Fig. 11.

Figure 12 represents an explanatory combination weaving model linking the migration alternatives R_1m_2 and R_1t_1, which in turn are represented on the left-hand side and on the right-hand side of the same figure, respectively. In particular, on the left-hand side of Fig. 12 the input model $\boxed{\text{m}}$ is linked by the weaving model in the middle to specify that the three `Person` instances will be migrated to instances of the metaclass `Client` defined in the new version of the metamodel $\boxed{\text{MM'}}$. On the right-hand side of Fig. 12 the element $\boxed{\text{T}}$ represents the transformation `Company2CRM`, which is linked with the variability model $\boxed{\text{TVWM}}$ to represent all the possible migration alternatives. The selected alternative will migrate the input pattern of the `Person2Worker` rule to use the `Employee` metaclass; the `Person2Client` rule will be deleted. Considered in isolation these are two possible migration alternatives leading to information loss (as demonstrated in Fig. 8). The combination R_1m_2 and R_1t_1 defined with the $\boxed{\text{Comb VWM}}$ shown in Fig. 12 will create for each `Person` instance a migrated `Client` element with a total information loss, since the chosen migration for $\boxed{\text{TVWM}}$ will match input `Employee` instances.

Figure 13 shows the specification of the (R_1m_3,R_1t_3) migration pair. According to the left-hand side of the shown combined weaving model, `Person` instances are migrated to `Employee` or `Client` ones, depending on the content of the `position` attribute. The transformation migration creates two different rules matching the two metaclasses (see the right-hand side of the same figure). By applying the shown combined weaving model, there will be no information loss for the `Person` instances available in the source model. The specification of combination weaving models like those shown in Fig. 12 and Fig. 13 is enabled by the conceived metamodel given in Fig. 14. In particular, a `CombinationModel` have references to the managed metamodel refactorings, and for each refactoring multiple pairs of migrations are possible. The number of alternatives in a single combination are related to multiple possible involved

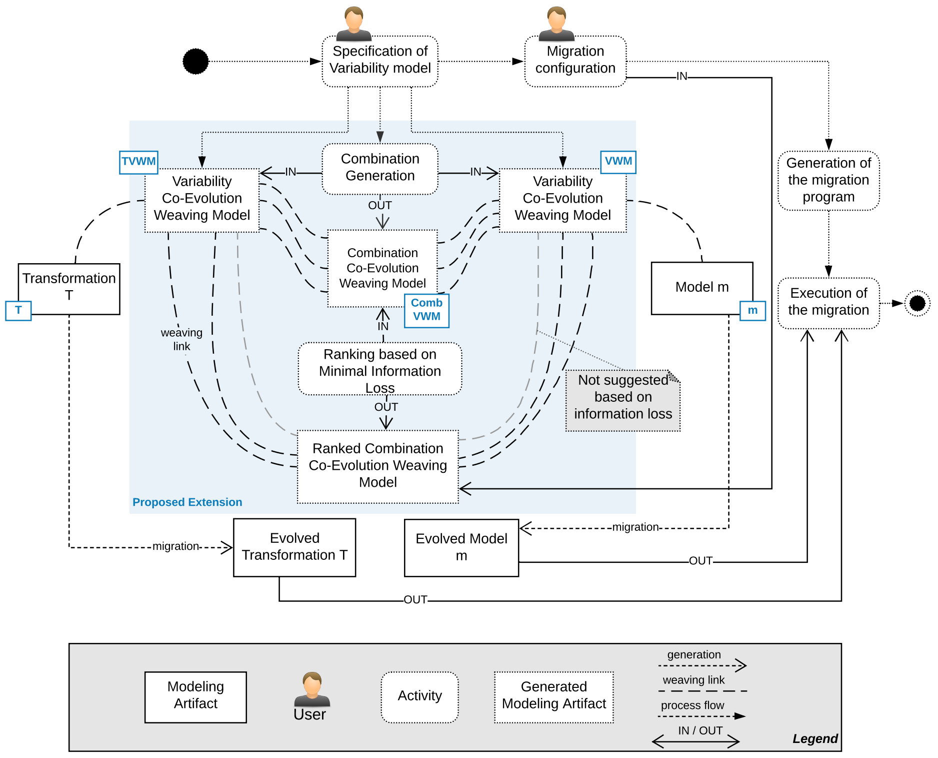
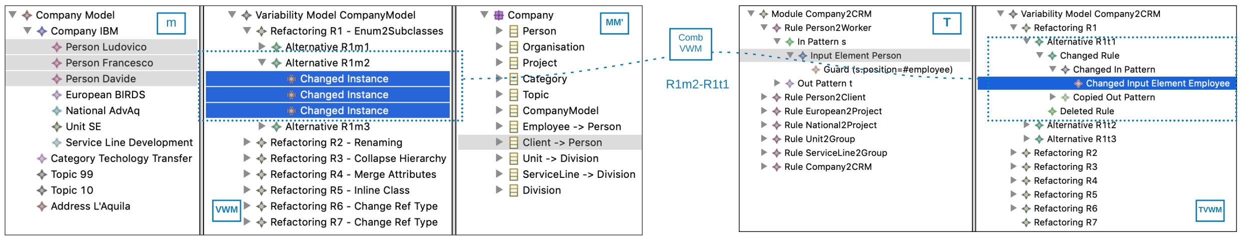

Specification of Variability model
Migration configuration
IN
Generation of the migration program
Execution of the migration
TVWM
Variability Co-Evolution Weaving Model
Combination Generation
IN
IN
VWM
Variability Co-Evolution Weaving Model
OUT
Combination Co-Evolution Weaving Model
Comb VWM
Transformation T
T
Model m
m
weaving link
IN
Ranking based on Minimal Information Loss
OUT
Not suggested based on information loss
Ranked Combination Co-Evolution Weaving Model
Proposed Extension
Evolved Transformation T
migration
Evolved Model m
migration
OUT
OUT
Modeling Artifact
User
Activity
Generated Modeling Artifact
generation
weaving link
process flow
IN / OUT
Legend

Figure 11 Proposed approach



```
▼ Company Model                              m
  ▼ Company IBM
      Person Ludovico
      Person Francesco
      Person Davide
      European BIRDS
      National AdvAq
      Unit SE
      Service Line Development
    Category Techology Transfer
    Topic 99
    Topic 10
    Address L'Aquila
```

```
▼ Variability Model CompanyModel
  ▼ Refactoring R1 - Enum2Subclasses
    ▶ Alternative R1m1
    ▼ Alternative R1m2
        Changed Instance
        Changed Instance
        Changed Instance
    ▶ Alternative R1m3
  ▶ Refactoring R2 - Renaming
  ▶ Refactoring R3 - Collapse Hierarchy
  ▶ Refactoring R4 - Merge Attributes
  ▶ Refactoring R5 - Inline Class
  ▶ Refactoring R6 - Change Ref Type
  ▶ Refactoring R7 - Change Ref Type
VWM
```

```
▼ Company                                    MM'
  ▶ Person
  ▶ Organisation
  ▶ Project
  ▶ Category
  ▶ Topic
  ▶ CompanyModel
  ▶ Employee -> Person
  ▶ Client -> Person
  ▶ Unit -> Division
  ▶ ServiceLine -> Division
  ▶ Division
```

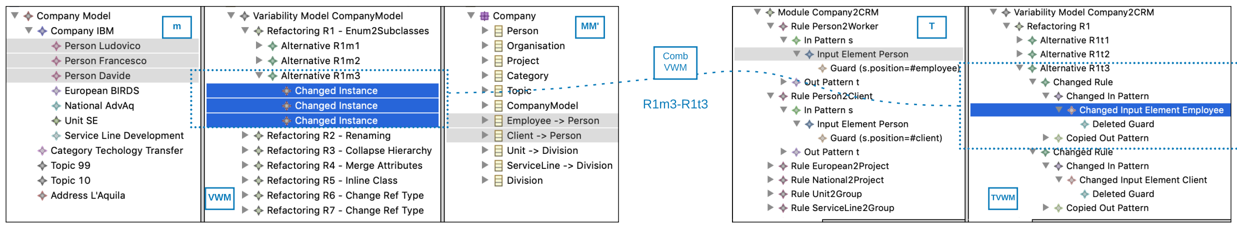
Comb VWM

R1m2-R1t1

```
▼ Module Company2CRM                          T
  ▼ Rule Person2Worker
    ▼ In Pattern s
      ▼ Input Element Person
          Guard (s.position=#employee)
    ▶ Out Pattern t
  ▶ Rule Person2Client
  ▶ Rule European2Project
  ▶ Rule National2Project
  ▶ Rule Unit2Group
  ▶ Rule ServiceLine2Group
  ▶ Rule Company2CRM
```

```
▼ Variability Model Company2CRM
  ▼ Refactoring R1
    ▼ Alternative R1t1
      ▼ Changed Rule
        ▼ Changed In Pattern
            Changed Input Element Employee
        ▶ Copied Out Pattern
        Deleted Rule
    ▶ Alternative R1t2
    ▶ Alternative R1t3
  ▶ Refactoring R2
  ▶ Refactoring R3
  ▶ Refactoring R4
  ▶ Refactoring R5
  ▶ Refactoring R6
    Refactoring R7                            TVWM
```

Figure 12 Specification of the (R_1m_2,R_1t_1) migration pair



```
▼ Company Model                              m
  ▼ Company IBM
      Person Ludovico
      Person Francesco
      Person Davide
      European BIRDS
      National AdvAq
      Unit SE
      Service Line Development
    Category Techology Transfer
    Topic 99
    Topic 10
    Address L'Aquila
```

```
▼ Variability Model CompanyModel
  ▼ Refactoring R1 - Enum2Subclasses
    ▶ Alternative R1m1
    ▶ Alternative R1m2
    ▼ Alternative R1m3
        Changed Instance
        Changed Instance
        Changed Instance
  ▶ Refactoring R2 - Renaming
  ▶ Refactoring R3 - Collapse Hierarchy
  ▶ Refactoring R4 - Merge Attributes
  ▶ Refactoring R5 - Inline Class
  ▶ Refactoring R6 - Change Ref Type
  ▶ Refactoring R7 - Change Ref Type
VWM
```

```
▼ Company                                    MM'
  ▶ Person
  ▶ Organisation
  ▶ Project
  ▶ Category
  ▶ Topic
  ▶ CompanyModel
  ▶ Employee -> Person
  ▶ Client -> Person
  ▶ Unit -> Division
  ▶ ServiceLine -> Division
  ▶ Division
```

Comb VWM

R1m3-R1t3

```
▼ Module Company2CRM                          T
  ▼ Rule Person2Worker
    ▼ In Pattern s
      ▼ Input Element Person
          Guard (s.position=#employee)
    ▶ Out Pattern t
  ▼ Rule Person2Client
    ▼ In Pattern s
      ▼ Input Element Person
          Guard (s.position=#client)
    ▶ Out Pattern t
  ▶ Rule European2Project
  ▶ Rule National2Project
  ▶ Rule Unit2Group
  ▶ Rule ServiceLine2Group
```

```
▼ Variability Model Company2CRM
  ▼ Refactoring R1
    ▶ Alternative R1t1
    ▶ Alternative R1t2
    ▼ Alternative R1t3
      ▼ Changed Rule
        ▼ Changed In Pattern
          ▼ Changed Input Element Employee
              Deleted Guard
        ▶ Copied Out Pattern
      ▼ Changed Rule
        ▼ Changed In Pattern
          ▼ Changed Input Element Client
              Deleted Guard
        ▶ Copied Out Pattern
TVWM
```

Figure 13 Specification of the (R_1m_3,R_1t_3) migration pair

artifacts, e.g., models and transformations.

Having two or more variability weaving models allows the generation of all the possible combinations for each metamodel refactoring. Such a generation is performed by a dedicated tool, which takes as input $\boxed{\text{VWM}}$ and $\boxed{\text{TVWM}}$ models and produces a corresponding $\boxed{\text{Comb VWM}}$ model. Such a generated model contains all the possible migration pairs. This model is then processed by the `Information Loss Ranker` that will rank all the combinations based on the information loss criteria.

The final result of the proposed process is shown to the user as depicted in Fig. 15: on the left-hand side the unordered generated combination weaving model is reported, in the middle the ranked combinations are shown as a model. A textual repre-

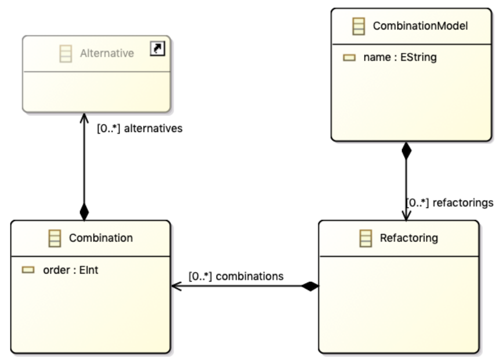


Figure 14 Fragment of the Combination metamodel

sentation of them are also provided as shown on the right-hand side of Fig. 15. Once the user selects the migration pair to be operated, the corresponding adaptation programs can be generated and applied by relying on the supporting infrastructure already presented in (Di Ruscio et al. 2016; Di Ruscio et al. 2017).

6. Tool support

The approach presented in the previous section has been fully implemented. In this section we make an overview of the developed tool by focusing on the `Combination Generation` and `Ranking based on Minimal Information Loss` phases shown in Fig. 11. The conceived supporting tool has been developed in Java by exploiting EMF technologies. Listing 5 shows a fragment of the method generating the combination model from input variability models related to the transformation (TVWM) and model (VWM) under analysis. The logic behind this method is quite simple: given a single metamodel refactoring and the two variability models, it creates a combination of them for each alternative pair (see line 19). Afterwards, the created collection of alternatives is sorted by a customized ordering method (see line 23) based on the information loss.

Listing 5 Fragment of the `createRankedCombinationModel` source code

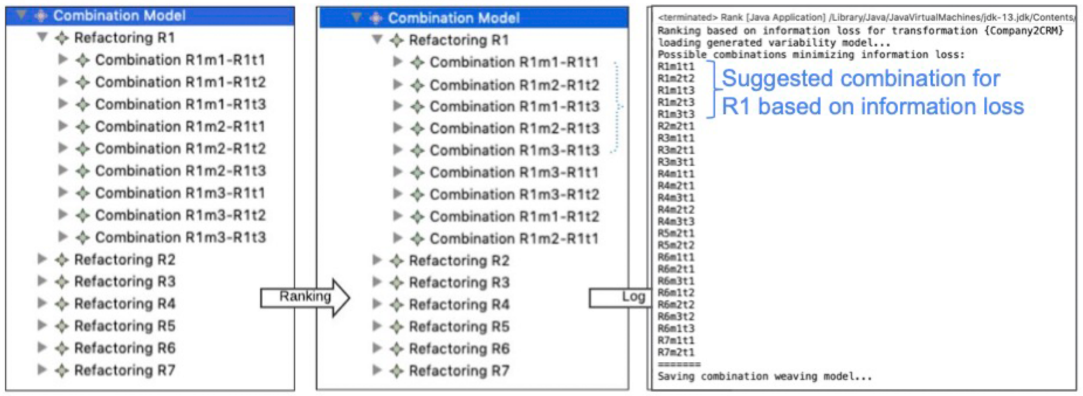
```
1  ...
2  private static void createRankedCombinationModel(VariabilityModel VWM,
       ↪ VariabilityModel TVWM) {
3         // Create Combination Model ROOT
4         CombinationModel combinationModel = COMBINATION_FACTORY.
       ↪ createCombinationModel();
5         //Create a list of Refactorings
6         List<Refactoring> refactorings = new ArrayList<Refactoring>();
7  
8         //Foreach refactoring in VWM
9         for (VariabilityWeavingMM.Refactoring modelRefactoring : VWM.
       ↪ getRefactorings()) {
10       //create a Refactoring
11       Refactoring refactoring = COMBINATION_FACTORY.createRefactoring();
12       refactoring.setName(modelRefactoring.getName());
13 
14                List<Combination> combinations = null;
15                //Foreach refactoring in TVWM
16                for (VariabilityWeavingMM.Refactoring
       ↪ transformationRefactoring : TVWM.getRefactorings()){
17                if(modelRefactoring.getName().equalsIgnoreCase(
       ↪ transformationRefactoring.getName())){
18                    //create Combinations for Refactoring R1 in VWM and R1
       ↪ .....Rx in TVWM
19                    combinations = createCombinations(modelRefactoring,
       ↪ transformationRefactoring);
20                }
21            }
22    //Sort Combination in Refactoring by custom combination comparator
23    combinations.sort(new CombinationComparator());
24    refactoring.getCombinations().addAll(combinations);
25    //Add new (Sorted) Refactoring IN CombinationModel.ranked.xmi
26    refactorings.add(refactoring);
27   }
28   combinationModel.getRefactorings().addAll(refactorings);
29   saveModel(combinationModel);
30 }
```

The ranking method shown in Listing 6 analyzes all the alternatives in the combination model and assigns a corresponding weight to each of them. For instance the alternative R_1m_1 contains three `ChangedInstance` elements referring to the *Person* instances that are assigned to the new type `Employee`. For this reason the ranker assign a positive weight (see promotion method at line 35 called at line 20) to the combination with R_1t_1 since this transformation alternative contains a `ChangedInPattern` element matching `Employee` instances. Indeed, this is considered to be a better solution than alternative ones e.g., R_1m_1 in combination with R_1t_2, which migrate the original transformation to match `Client` elements. Thus, the combination (R_1m_1,R_1t_2) is penalized since three migrated instances will not be matched according to the penalization method shown at line 39 and called in line 23. In other words, the ranker goes through all the possible alternative by querying the variability models looking for patterns to be penalized or promoted. At the end of the process, every combination will have a corresponding weight assigned by enabling their final ordering.

Concerning the execution performance of the developed tools, the proposed ranking algorithm takes ≈2 seconds to be executed on models having sizes, which are similar to those of the case study models. According to the performed experiments, the execution time seems to increase linearly with the size of the input models. However, we expect that including the management of constructs that are not covered at the moment, e.g., OCL expressions, can increase the overall execution times of the approach.

Listing 6 Fragment of the `evaluateAlternatives` source code

```
1  public static int evaluateAlternatives(Alternative RxMy, Alternative RxTy)
       ↪  {
2    int rank = 0;
3    TreeIterator<EObject> eAllContents = RxMy.eAllContents();
4    while (eAllContents.hasNext()) {
5     EObject next = eAllContents.next();
6  
7     if(next instanceof ChangedInstance) {
8      ChangedInstance changeElement = (ChangedInstance) next;
9      EObject referencedObject = changeElement.getNewtype();
10 
11     //Check types match with RxTy alternative
12     TreeIterator<EObject> RxTyeAllContents = RxTy.eAllContents();
13     while (RxTyeAllContents.hasNext()) {
14      EObject innerNext = RxTyeAllContents.next();
15 
16      if(innerNext instanceof ChangedInputElement) {
17       ChangedInputElement innerChangeElement = (ChangedInputElement)
         ↪ innerNext;
18       //If the class is the same then PROMOTE it
19       if(referencedObject.toString().contains(innerChangeElement.
         ↪ getExpression())) {
20        rank = promote(rank);
21       }else {
22        //Otherwise PENALIZE it
23        rank = penalize(rank);
```



```
Combination Model
  Refactoring R1
    Combination R1m1-R1t1
    Combination R1m1-R1t2
    Combination R1m1-R1t3
    Combination R1m2-R1t1
    Combination R1m2-R1t2
    Combination R1m2-R1t3
    Combination R1m3-R1t1
    Combination R1m3-R1t2
    Combination R1m3-R1t3
  Refactoring R2
  Refactoring R3
  Refactoring R4
  Refactoring R5
  Refactoring R6
  Refactoring R7
```

```
Combination Model
  Refactoring R1
    Combination R1m1-R1t1
    Combination R1m2-R1t2
    Combination R1m1-R1t3
    Combination R1m2-R1t3
    Combination R1m3-R1t3
    Combination R1m3-R1t1
    Combination R1m3-R1t2
    Combination R1m1-R1t2
    Combination R1m2-R1t1
  Refactoring R2
  Refactoring R3
  Refactoring R4
  Refactoring R5
  Refactoring R6
  Refactoring R7
```

```
<terminated> Rank [Java Application] /Library/Java/JavaVirtualMachines/jdk-13.jdk/Contents/
Ranking based on information loss for transformation {Company2CRM}
loading generated variability model...
Possible combinations minimizing information loss:
R1m1t1
R1m2t2
R1m1t3
R1m2t3
R1m3t3
R2m2t1
R3m1t1
R3m2t1
R3m3t1
R4m1t1
R4m2t1
R4m3t1
R4m2t2
R4m3t3
R5m2t1
R5m2t2
R6m1t1
R6m2t1
R6m3t1
R6m1t2
R6m2t2
R6m3t2
R6m1t3
R7m1t1
R7m2t1
========
Saving combination weaving model...
```

Figure 15 Simple generated combination model

```
        }
      }
      ...
      //Other checks here
      ...
    }
  }
 }
 return rank;
}

private static int promote(int rank) {
 return rank + 1;
}

private static int penalize(int rank) {
 if(rank > 0) {
  return rank - 1;
 }else {
  return 0;
 }
}
```

7. Evaluation

In this section we present the experiments that we have performed to assess the accuracy of the proposed approach with the aim of answering the following research question:

> ***RQ**: Given a metamodeling ecosystem, does the proposed approach correctly rank pairs of possible model and transformation migrations with respect to the induced information loss in the response of metamodel refactorings?*

To answer such research question a dedicated evaluation process has been implemented as descried in Section 7.1. The analysis of the obtained results are discussed in Section 7.2.

7.1. Evaluation process

The performed evaluation has been done by considering the model `CompanyModel.xmi`[2] and the transformation `Company2CRM`[3] both introduced in Sec. 2. In particular, the refactorings R1–R7 described in the previous section, have been singularly applied on the `Company` metamodel[4] and all the possible migration alternatives of `CompanyModel.xmi` and `Company2CRM` are automatically generated as described in Sec. 4. For the sake of presentation, Figure 16 shows one of the refactoring application and the process has been iterated for each applied refactoring, and for each possible model and transformation migration pair. For instance, refactoring R1 gives place to 3 model and 3 transformation alternatives; each combination of them has been given to the `Information Loss Calculator` component, which is able to calculate the information loss that might be induced by executing the migration pair under analysis if executed on the real input models. The evaluation process has been fully implemented and is publicly available.[5]

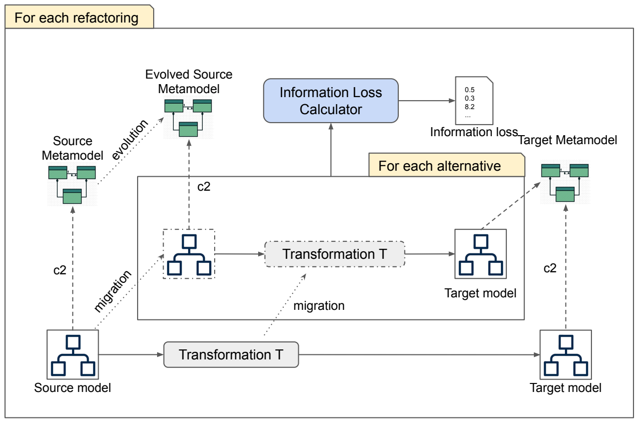


Figure 16 Evaluation process

7.2. Analysis of the results

The output of the information loss calculator applied to all the combinations of model and transformation migrations is shown in Table 1. The cells in bold represent the combinations with the lowest induced information loss (thus preferable). Among them, it is possible to have combinations inducing the same information loss value. For instance, in the case of refactoring `R1`, different combinations induce the lowest information loss value, i.e., 12.5. In the case of refactoring `R2`, only the migration pair (R_2m_1,R_2t_1) induces the lowest information loss.

[2] `https://github.com/gssi/variabilityCo-evoIL/blob/main/m/CompanyModel.xmi`

[3] `https://github.com/gssi/variabilityCo-evoIL/tree/main/t`

[4] `https://github.com/gssi/variabilityCo-evoIL/blob/main/mm/Company.ecore`

[5] `https://github.com/gssi/QualityEvaluationIL`

Concerning the refactoring R7, there is no need of migrating the original transformation, which is not affected by the operated metamodel refactoring, i.e. a non-breaking change w.r.t. the transformation. Thus, only the input model has to be migrated and it can be done in two different ways as shown in Table 1. For both options, the induced information loss is the same.

Table 1 Calculated information loss values

		Transformation migration		
Refactoring R1		R_1t_1	R_1t_2	R_1t_3
Model migration	R_1m_1	**12.5**	14	**12.5**
	R_1m_2	14	**12.5**	**12.5**
	R_1m_3	13	13.5	**12.5**
Refactoring R2		R_2t_1	R_2t_2	
Model migration	R_2m_1	14.5	18.5	
	R_2m_2	**11**	11.5	
Refactoring R3		R_3t_1	R_3t_3	R_3t_3
Model migration	R_3m_1	**13.5**	14.5	18
	R_3m_2	**13.5**	14	16
	R_1m_3	**13.5**	14	16
Refactoring R4		R_4t_1	R_4t_2	R_4t_4
Model migration	R_4m_1	**14.5**	15.25	15.25
	R_4m_2	**14.5**	**14.5**	15.25
	R_4m_3	**14.5**	15.25	**14.5**
Refactoring R5		R_5t_1	R_5t_2	
Model migration	R_5m_1	13	13.5	
	R_5m_2	**12.5**	**12.5**	
Refactoring R6		R_6t_1	R_6t_2	R_6t_3
Model migration	R_6m_1	**15.5**	**15.5**	**15.5**
	R_6m_2	**15.5**	**15.5**	16.5
	R_6m_3	**15.5**	**15.5**	16.5
Refactoring R7		t		
Model migration	R_7m_1	**15.5**		
	R_7m_2	**15.5**		

Table 2 shows a summary of the suggested migration pairs for each operated metamodel refactoring. The table represents only those pairs that induce the lowest information loss. Each refactoring has been manually analyzed in order to check if the suggested recommendations were correct with respect to the induced information loss. Such a phase has been done by executing each transformation migration on each obtained migration model. For each execution, the corresponding information loss has been measured on the generated CRM model and at the end the lowest values have been compared with those calculated by the presented approach. For all the cases, the suggested migration pairs were confirmed, meaning that the proposed solution was the right one. Concerning the last refactoring, it cannot be considered as confirmed, as there were no combinations to be evaluated as only model migrations were needed in response of the R7 refactoring.

8. Discussion

The main strengths of the approach proposed in this paper are related to the possibility of identifying combinations of model and transformation migrations that can induce a lowest value of information loss. However, the approach as proposed in this paper can be enhanced in different directions as discussed in the following:

Additional criteria for selecting model and transformation migration pairs: Information loss is one of the possible criteria that can be employed to support modelers when they have to select the right combination of model and transformation migrations. Further techniques might be explored including the adoption of static analysis of model transformations. Specifically in terms of information loss, even though removing affected model instances or transformation rules can solve the inconsistency problems caused by metamodel evolutions, by definition it gives place to information loss. In fact also in our example a combination, specifically in R2 induces that if the modeler picks (R_2m_2, R_2t_1) or (R_2m_2, R_2t_2) combinations will result in no information loss (with respect to that refactoring), because the alternative m_2 removes the affected instances to restore conformance, so it would not be preferable to select the alternatives using removal, because in combination will result in not loosing information, but for definition it is removing something from the models. This case is relevant when for some reason we have multiple solution at the same rank position, but a solution removes something from the model by definition. For this reason we intend to further extend the approach to give a weight to the internal instances of the model, i.e. semantic importance, or give an internal rank to the alternatives of migration.

Management of parallel dependent metamodel evolutions: The metamodel evolutions included in this paper are parallel independent, meaning that they act on different elements, for this reason it has been possible to apply the methodology for the evaluation of the approach. Otherwise, the application of the proposed technique would be dependant on the execution order of the migrations. The management of parallel dependent metamodel evolutions, and thus of the consequent issues related to the management of migration conflicts, deserves further investigation.

Table 2 Summary of the suggested migration pairs

Refactoring	Suggested migration pair	Confirmed
R1	(R_1m_1, R_1t_1)	✓
	(R_1m_1, R_1t_3)	✓
	(R_1m_2, R_1t_2)	✓
	(R_1m_2, R_1t_3)	✓
	(R_1m_3, R_1t_3)	✓
R2	(R_2m_1, R_2t_1)	✓
R3	(R_3m_1, R_3t_1)	✓
	(R_3m_1, R_3t_2)	✓
R4	(R_4m_1, R_4t_1)	✓
R5	(R_5m_2, R_5t_1)	✓
	(R_5m_2, R_5t_2)	✓
R6	(R_6m_1, R_6t_2)	✓
	(R_6m_2, R_6t_2)	✓
	(R_6m_3, R_6t_2)	✓
R7	R_7m_1 R_7m_2	-

Transformation complexity: The transformations that have been considered in the approach are developed in the ATL language, a complex rule-based transformation language offering also imperative constructs and additional concepts that we do not consider at the moment. This can further increase the complexity in representing the transformations as migration alternatives. Specifically, our approach does not consider OCL constraints that can be applied as filter of the transformation rules. This aspect is listed in this category since the transformation should be dynamically analyzed in relation to the included OCL expressions before detecting the ranking of the proposed solutions. For this reason, we list this aspect as a possible limitation, even if the approach still applies if the injection of the transformation into the weaving model can be improved to consider also these possible cases.

Experimentation and evaluation: To evaluate the approach we have considered the explanatory ecosystem, even though more than 40 migration pairs have been analyzed to calculate the induced information loss. To better assess the validity of the proposed approach it is necessary to consider an extended data set including more metamodels, refactorings, models and transformations. We specify that the migration alternatives presented in Section 3 have been chosen by inspecting the most applied strategies in literature and in our experience, but the alternatives could be more extended and this aspect should be investigated in the future.

Technological Specificity: The proposed approach is generic since it relies on a weaving-based representation of migration variability (Di Ruscio et al. 2016; Di Ruscio et al. 2017). This means that even if the approach has been developed and tested in the EMF technological space, it is possible to apply it in other technologies by implementing the management of the proposed variability weaving models in the technology of interest.

Required Effort: The proposed approach is based on a set of artifacts and semi-automatic activities that need to be completed with human intervention. In order to enable the final ranking of the found solutions, a set of artifacts need to be produced and inspected, i.e., the variability model and the configuration for the migration to be operated. To give some numbers, the case study presented in Section 2 took $\approx$20 minutes for the configuration. Part of the effort will be reduced when the user becomes familiar with the tooling and with the provided representations, but we are aware that if we consider the complete syntax of the transformation language (even if the production of the weaving is automated) it can be difficult to be represented in terms of weaving models. We plan to investigate this aspect in future work, with the intent of reducing the complexity of the entire generation / ranking process, and improving the usability of the provided tools to make easier the specification of the needed input models.

9. Related Work

Coupled Evolution is an extensively investigated topic by the model-driven engineering community. Our previous work in (Di Ruscio et al. 2012) discusses the different relations occurring in a typical metamodeling ecosystem among the metamodel and the related artifacts, and identifies the commonalities supporting the definition of a unifying and comprehensive adaptation process. Multiple approaches have been presented over the years dealing with the evolution of metamodel and migration of models. COPE (Herrmannsdoerfer et al. 2009) is an approach to specify the coupled evolution of metamodels and models in which a language is evolved by incrementally composing modular coupled transformations that adapt the metamodel and specify the corresponding model migrations. Flock (Rose et al. 2010) is a coupled evolution approach built on top of the Epsilon Framework, offering a model-to-model transformation language tailored for model migration. Wimmer et al. (Wimmer et al. 2010) instead of describing the co-evolution of models as a transformation between two metamodels, they employ existing inplace transformation languages. Thus, the prerequisite is to merge the initial and the revised metamodel ensuring that the initial as well as the revised model conform to the merged metamodel, enabling the employment of inplace transformations for initializing new metamodel elements. At the end, a check-out transformation eliminates model elements which are no longer covered by the revised metamodel. Other works present mechanisms to define and execute coupled evolutions of metamodels and instance models (Krause et al. 2013) or transformations (Rutle et al. 2018a, 2020) based on graph transformations. These approaches treat coupled evolutions as dynamically typed graph transformation rules.

In (Herrmannsdoerfer et al. 2010) authors present a catalogue of coupled operators for an operator-based approach that is based on a literature survey. Also transformations (Mendez et al. 2010; Wagelaar et al. 2012; García et al. 2012; Rutle et al. 2018b) and editors (Di Ruscio et al. 2011; Di Rocco et al. 2014) migration in response to metamodel evolutions has been a topic already investigated. One of the first and only attempts considering multiple migration alternatives and ranking them is proposed in (Schönböck et al. 2014). The authors employ logic programming to generate a set of ranked solutions for model migrations based on the formalization of the conformance relationship. Information loss is considered in ranking the solutions but only at the model level, excluding transformations. All these works concentrate the effort in creating tools or languages for expressing co-evolution strategies in specific cases ignoring the possible multiple resolutions or demanding it to the modeler. To the best of our knowledge, the presented approach is the first attempt in dealing with parallel couple evolution, where artifacts are considered in combination, instead of treating the corrupted artifact in a single batch of migration.

A different application field can be seen when traceability links are used to establish correspondences among requirements and the different parts of a software system. The authors in (Riebisch 2004) use feature models as an intermediate element for linking requirements to design models. In this way changes to the requirements can be reflected immediately to feature model elements in order to activate the resolution of possible inconsistencies. This approach works in requirement engineering, whereas the presented approach in coupled evolution. Another attempt in the same direction has been made by (Gamez & Fuentes 2011). They propose a model-driven approach to propagate changes made in an evolved feature model into existing configurations. Also the authors in (Guo et al. 2012) present an approach to deal with inconsistencies in feature model evolutions. These papers present a different level of application, with the same tool we use, i.e. feature models, in order to propagate the applied changes to other artifacts.

The tool called MoDisco (Bruneliere et al. 2014), among its features, presents how to avoid information loss of models but in a different context w.r.t. our approach, i.e. Model Driven Reverse Engineering. Unlike our system it focuses on retrieving as much information as possible from a legacy system that needs to be updated. In (Fritsche et al. 2020), in the area of model synchronization, the authors focus on avoiding information loss when, through Triple Graph Grammars (TGGs), new rules for synchronizing models are automatically created. The purpose of a TGG is to define a consistent relationship between pairs of models in a rule-based manner by defining traces between their elements. Given a TGG, its rules can be automatically operationalized into the source and forward rules. Loss of information can occur when there are elimination rules that could result in the loss of information when applied. Compared to our contribution, this work focuses on finding solutions to these rules' definition to avoid such information loss but does not rely on evaluating existing models and, consequently, assessing how much information is lost when a transformation between models is performed.

10. Conclusion and Future Work

This paper proposes an approach for visualizing and representing possible combinations of migration alternatives for corrupted artifacts using weaving models in response to metamodel evolutions. The presented approach is supported by a tool which also refines the number of possible migration pairs by relying on the notion of *information loss*. Thus, modelers are provided with an ordered list of migration pairs with the aim of supporting them in the selection and generation of the corresponding programs to be executed on the affected artefacts. In this work we focused on models and model transformations even though the approach can be extended to add other types of artifacts. In this respect, as future work we plan to introduce the management of different input models and of additional kinds of modeling artifacts including code generators and editors, and to apply the approach with further metamodel refactorings. This aspect requires additional effort first to translate the different kinds of artifacts in terms of weaving models for traceability purposes, and subsequently to enable the link with the information loss w.r.t. the given input models. For instance, if we consider a code generator we would need to investigate whether the generator template under analysis uses modeling concepts affected by the considered metamodel evolution, and then to establish the possible relations with the input models.

Acknowledgments

The research described in this paper has been partially supported by the AIDOaRt Project, EU H2020-ECSEL European Programme, https://www.aidoart.eu/

References

Basciani, F., D'Emidio, M., Di Ruscio, D., Frigioni, D., Iovino, L., & Pierantonio, A. (2018). Automated selection of optimal model transformation chains via shortest-path algorithms. *IEEE Transactions on Software Engineering*.

Bettini, L., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2019). Quality-driven detection and resolution of metamodel smells. *IEEE Access*, *7*, 16364-16376. doi: 10.1109/ACCESS.2019.2891357

Bézivin, J. (2005, May). On the unification power of models. *Software & Systems Modeling*, *4*(2), 171–188. doi: 10.1007/s10270-005-0079-0

Bézivin, J., Jouault, F., Rosenthal, P., & Valduriez, P. (2005). Modeling in the large and modeling in the small. In U. Aßmann, M. Aksit, & A. Rensink (Eds.), *Model driven architecture* (pp. 33–46). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bruneliere, H., Cabot, J., Dupé, G., & Madiot, F. (2014). Modisco: A model driven reverse engineering framework. *Information and Software Technology*, *56*(8), 1012–1032.

Di Ruscio, D., Etzlstorfer, J., Iovino, L., Pierantonio, A., & Schwinger, W. (2017). A feature-based approach for variability exploration and resolution in model transformation migration. In A. Anjorin & H. Espinoza (Eds.), *Ecmfa* (Vol. 10376, pp. 71–89). Springer. doi: 10.1007/978-3-319-61482-3_5

Di Ruscio, D., Iovino, L., & Pierantonio, A. (2012). Evolutionary togetherness: How to manage coupled evolution in metamodeling ecosystems. In H. Ehrig, G. Engels, H. Kreowski, & G. Rozenberg (Eds.), *Graph transformations - 6th international conference, ICGT 2012, bremen, germany, september 24-29, 2012. proceedings* (Vol. 7562, pp. 20–37). Springer. doi: 10.1007/978-3-642-33654-6_2

Di Rocco, J., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2014). Dealing with the coupled evolution of metamodels and model-to-text transformations. In *Me workshop @models* (pp. 22–31).

Di Ruscio, D., Etzlstorfer, J., Iovino, L., Pierantonio, A., & Schwinger, W. (2016). Supporting variability exploration and resolution during model migration. In *Ecmfa*.

Di Ruscio, D., Iovino, L., & Pierantonio, A. (2012). Evolutionary Togetherness: How to Manage Coupled Evolution in Metamodeling Ecosystems. In *Icgt* (Vol. 7562).

Di Ruscio, D., Iovino, L., & Pierantonio, A. (2013). Managing the coupled evolution of metamodels and textual concrete syntax specifications. In *2013 39th euromicro conference on software engineering and advanced applications* (pp. 114–121).

Di Ruscio, D., Lämmel, R., & Pierantonio, A. (2011). Automated co-evolution of gmf editor models. In B. Malloy, S. Staab, & M. van den Brand (Eds.), *Software language engineering* (pp. 143–162). Berlin, Heidelberg: Springer Berlin Heidelberg.

Falzone, E., & Bernaschina, C. (2018). Model Based Rapid Prototyping and Evolution of Web Application. In T. Mikkonen, R. Klamma, & J. Hernández (Eds.), *Web Engineering* (Vol. 10845, pp. 496–500). Cham: Springer International

Publishing. doi: 10.1007/978-3-319-91662-0_43

Fritsche, L., Kosiol, J., Schürr, A., & Taentzer, G. (2020). Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars. *International Journal on Software Tools for Technology Transfer*, 1–34.

Gamez, N., & Fuentes, L. (2011). Software product line evolution with cardinality-based feature models. In K. Schmid (Ed.), *Top productivity through software reuse* (pp. 102–118). Berlin, Heidelberg: Springer Berlin Heidelberg.

Garcés, K., Vara, J. M., Jouault, F., & Marcos, E. (2013). Adapting transformations to metamodel changes via external transformation composition. *Software & Systems Modeling*.

García, J., Diaz, O., & Azanza, M. (2012). Model transformation co-evolution: A semi-automatic approach. In *International conference on software language engineering* (pp. 144–163).

Guo, J., Wang, Y., Trinidad, P., & Benavides, D. (2012). Consistency maintenance for evolving feature models. *Expert Systems with Applications*, *39*(5), 4987 - 4998. doi: https://doi.org/10.1016/j.eswa.2011.10.014

Hebig, R., Khelladi, D. E., & Bendraou, R. (2016). Approaches to co-evolution of metamodels and models: A survey. *IEEE Transactions on Software Engineering*, *43*(5), 396–414.

Herrmannsdoerfer, M., Benz, S., & Juergens, E. (2009). Cope - automating coupled evolution of metamodels and models. In *Proc. of ecoop* (pp. 52–76). Springer.

Herrmannsdoerfer, M., Vermolen, S. D., & Wachsmuth, G. (2010). An extensive catalog of operators for the coupled evolution of metamodels and models. In *International conference on software language engineering* (pp. 163–182).

Hutchinson, J., Whittle, J., & Rouncefield, M. (2014). Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, *89*, 144–161.

Iovino, L., Pierantonio, A., & Malavolta, I. (2012, October). On the Impact Significance of Metamodel Evolution in MDE. *JoT*, *11*(3), 3:1-33.

Kessentini, W., Sahraoui, H., & Wimmer, M. (2018). Automated Co-evolution of Metamodels and Transformation Rules: A Search-Based Approach. In T. E. Colanzi & P. McMinn (Eds.), *Search-Based Software Engineering* (Vol. 11036, pp. 229–245). Cham: Springer International Publishing. doi: 10.1007/978-3-319-99241-9_12

Kessentini, W., Wimmer, M., & Sahraoui, H. (2018, October). Integrating the Designer in-the-loop for Metamodel/Model Co-Evolution via Interactive Computational Search. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (pp. 101–111). Copenhagen Denmark: ACM. doi: 10.1145/3239372.3239375

Krause, C., Dyck, J., & Giese, H. (2013). Metamodel-specific coupled evolution based on dynamically typed graph transformations. In K. Duddy & G. Kappel (Eds.), *Theory and practice of model transformations* (pp. 76–91). Berlin, Heidelberg: Springer Berlin Heidelberg.

Mendez, D., Etien, A., Muller, A., & Casallas, R. (2010). Towards transformation migration after metamodel evolution. *Model and Evolution Wokshop*.

Riebisch, M. (2004). Supporting evolutionary development by feature models and traceability links. In *Proceedings. 11th ieee international conference and workshop on the engineering of computer-based systems, 2004.* (p. 370-377).

Rose, L. M., Kolovos, D. S., Paige, R. F., & Polack, F. A. (2010). Model Migration with Epsilon Flock. In *Theory and practice of model transformations - 3rd international conference, ICMT* (Vol. 6142, pp. 184–198). Springer.

Rutle, A., Iovino, L., König, H., & Diskin, Z. (2018a). Automatic transformation co-evolution using traceability models and graph transformation. In *European conference on modelling foundations and applications* (pp. 80–96).

Rutle, A., Iovino, L., König, H., & Diskin, Z. (2018b). Automatic transformation co-evolution using traceability models and graph transformation. In A. Pierantonio & S. Trujillo (Eds.), *Modelling foundations and applications* (pp. 80–96). Cham: Springer International Publishing.

Rutle, A., Iovino, L., König, H., & Diskin, Z. (2020). A query-retyping approach to model transformation co-evolution. *Software and Systems Modeling*, *19*(5), 1107–1138. doi: 10.1007/s10270-020-00805-6

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

Schönböck, J., Kusel, A., Etzlstorfer, J., Kapsammer, E., Schwinger, W., Wimmer, M., & Wischenbart, M. (2014). CARE – A Constraint-Based Approach for Re-Establishing Conformance-Relationships. In *Proc. of the apccm.*

Thompson, N., & Platt, R. (2015, 01). The evolution of uml. In (p. 348-353).

Vallecillo, A. (2014, 12). On the industrial adoption of model driven engineering. is your company ready for mde? *International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)*, *1*, 52-68.

Wagelaar, D., Iovino, L., Di Ruscio, D., & Pierantonio, A. (2012). Translational semantics of a co-evolution specific language with the EMF transformation virtual machine. In *Theory and practice of model transformations - 5th international conference, ICMT* (Vol. 7303). Springer.

Wimmer, M., Kusel, A., Schönböck, J., Retschitzegger, W., Schwinger, W., & Kappel, G. (2010). On using inplace transformations for model co-evolution. In *Proc. 2nd int. workshop model transformation with atl* (Vol. 711, pp. 65–78).

About the authors

Francesco Basciani is postdoc researcher at the Università degli Studi dell'Aquila (Italy).You can contact him at francesco.basciani@univaq.it.

Davide Di Ruscio is professor at the Università degli Studi dell'Aquila (Italy). You can contact him at davide.diruscio@univaq.it.

Ludovico Iovino is assistant professor at the Gran Sasso Science Institute (Italy). You can contact him at ludovico.iovino@gssi.it.

Alfonso Pierantonio is full professor at the Università degli Studi dell'Aquila (Italy). You can contact him at alfonso.pierantonio@univaq.it or visit `http://pieranton.io`.