Adapting Models in Metamodels Composition Processes

Adaptación de modelos de procesos de composición a partir de metamodelos

Héctor Flórez*

Fecha de recepción: 11 de marzo de 2013
Fecha de aprobación: 30 de abril de 2013

Abstract

In Model Driven Engineering (MDE) approaches, metamodels can change after the creation of conformant models. Moreover, changes applied on one metamodel can be result of a composition process. When metamodels change, model conformity can be broken. Once the conformity is broken, the model is unuseful and it is not possible to regain the conformity with the composite metamodel. This paper presents a proposal to solve models adaptation through a Domain Speci c Language (DSL). This DSL is used by metamodelers who are the people that know the domains abstracted by several metamodels, and know how to combine those meta-models in order to generate the composite metamodel. In addition, the DSL allows metamodelers to include the solu-tion for conformant models adaptation.

Keywords

Model adaptation, metamodel composition, model driven engineering.

* Universidad Distrital Francisco José de Caldas Bogotá, Colombia. Correo electrónico: haflorezf@udistrital.edu.co

1. Introduction

Modeling has an important role in developing software systems because it provides means to concepts abstracted in a specific domain [15]. One model is a simplification of a system with an intended goal [2, 24]. In addition, in MDE approaches metamodels are used to abstract the concepts of a specific domain of information, and it is usually built by one metamodeler, who is the person that knows the do-main [8], and models, which represent a specific case in the domain [7], must conform to the correspondent metamodel [2].

Continually, domains changes and those changes can be the result of a composition process because this kind of process allows to reuse several concepts from several metamodels [5, 19]. However, if one metamodel changes, the conformant models lose the conformity. Consequently, it is necessary to modify the model in order to regain the conformity with the composite metamodel.

Nevertheless, it is not possible to modify the conformant models automatically in order to regain the conformity, be-cause the composite metamodel could have new elements that require additional information or the composition process could delete some elements from the original metamodel [8].

This proposal presents a solution strategy for models adaptation after a composition process. This solution is based on one Domain Specific Language (DSL) that has instructions to compose metamodels and to adapt conformant models. The metamodeler, who knows the domain, defines the changes on the metamodel and for each one of them, he/she also defines the changes to be applied for each instance related with the change on the metamodel.

The rest of the paper is structured as follows. Section 2 describes the metamodel composition problem. Section 3 presents the model adaptation problem. Section 4 presents the solution strategy. Section 5 presents the proposed languages for metamodels composition and models adaptation. Section 6 presents the proposal engine focusing in the com-position engine and the adaptation engine. Finally, section 7 presents the conclusions.

2. Metamodels composition

In MDE, metamodels composition is necessary for several reasons such us reusability, scalability, e ectiveness, among others [22, 26]. When a new domain is needed to be abstracted by a metamodel using an MDE approach, several previously constructed domains could abstract some elements that the new domain needs to be included [6]. Consequently, the e ort in the construction process of the new metamodel can be reduced as much as possible getting advantage of the e orts invested in the domains taken through the correspondent metamodels.

A metamodel composition strategy aims to support the construction of complex metamodels using atomic transformations [26]. There are some processes for metamodel com-position: 1) matching elements, 2) metamodel merge, and 3) class refinement. Matching models is a process used to identify diferent views of the same concept, in order to unify those several equivalent concepts in one composite concept [9]. Metamodel merge combines several concepts creating a new one in order to avoid collisions between the elements described in two diferent metamodels used for the metamodel composition process [5]. Class refinement is used to add de-tails in one single element that has not been composed with other elements [6].

3. Model adaptation

The evolution is a common event in the life cycle of a meta-model. This phenomenon happens when the models of the information are created by humans [2]. Metamodels evolution can be performed

by metamodels composition. The main reason why metamodels evolve is that the metamodel may be incomplete. In this case the evolution of a metamodel is driven by the need of fixing it to become more complete [14, 20]. This means that although all the elements of the information domain are supposed to be represented in the metamodel some concepts may be missing. In this case the evolution of a metamodel is driven by the need of fixing it to become more complete. Usually, when dealing with more than one information domain, there is the need to expand one metamodel through composition processes in order to be able to add information from other domains.

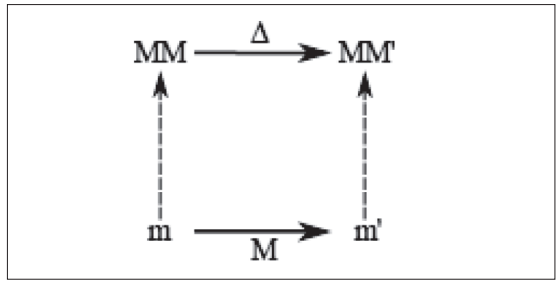
Changes on one metamodel impact all conformant models. The problem is presented when an element from the meta-model changes, and the model does not change breaking the conformity of the dependent model [8].

There are two main consequences when the conformity is lost in models created using Eclipse Modeling Framework (EMF) [4, 29]. First, in almost all cases, the model is not visible with some EMF graphic tool anymore. Second, although the model can be draw by a tool, the model is not valid because it does not satisfy the definition specified in the correspondent metamodel.

The process of adaptation is not always automatic or deductible. Sometimes, an input from the user is needed when there are changes that require creating new elements or redefining existing elements. In those cases, the model has to change in different ways keeping conformity with the evolved metamodel.

Figure 1, illustrates the metamodel evolution and models adaptation problem. After evolution of a metamodel MM into MM', the goal is to adapt the model m that conforms to MM, to m' that conforms to MM', by creating an appropriate adapting migration M [8, 23].

Figure 1. The metamodel evolution and model adap-tation illustrated



Reference: investigation.

There are some approaches that solve the models adaptation problem focusing in two main aspects: 1) identifying the differences between the original and the evolved metamodel using a declarative evolution specification to de ne a difference metamodel which can be calculated from identified changes in the metamodel [3], and 2) making the modifications on the model in order to regain the conformity with the updated metamodel [3] by a sequence of atomic operations where each operation is applied on metamodel and model level [1, 11, 17, 18].

Possible changes in metamodels can be classified as 1) non-breaking (NB), which are changes that have no impact on the model (e.g. increase the upper bound of one existing attribute); 2) breaking and automatically resolvable (BAR), which are the changes that have impact on the model, yet can be resolved automatically (e.g. rename an existing at-tribute); and 3) breaking and not automatically resolvable (BNAR), which are the changes that cannot be inferred, so need additional information that is provided by the modeler in order to be fulfilled (e.g. create a new attribute) [1, 10, 23, 25, 27, 28]. Having identified the changes each approach re-solves the first two categories of changes automatically using different frameworks for model transformation. To address the last category, some approaches take advantage of the user assistance to coevolve the models [1]. Some approaches for models adaptation are the following:

1. Becker et al. [1] propose an approach to address BNAR changes through a framework for assisting the user in the definition of model coevolution when a change of this category is found.

2. Cicchetti et al. [3] propose to classify the changes in atomic changes and de ne the process of adaptation. Then, create a differential metamodel with the identified changes, and it is classified in two new metamodels. If there are relations between the two metamodels, the adaptation is done using user intervention.

3. Herrmannsdoerfer et al. [16] approach the coevolution through the proposal called COPE that is a language to satisfy two requirements: 1) reuse of recurring migration knowledge and 2) expressiveness to support domain specific migrations.

4. Garces et al. [10] approach the coevolution through ATG (Adaptation Transformation Generation) that is a semiautomatic approach to generate an executable model adaptation transformation generating the adaptation transformation.

5. Florez et al. [8] approach the coevolution through ASI-MOV that is an approach to solve metamodel evolution and models coevolution through two DSLs. The first DSL allows to specify the changes on the metamodel and the second DSL allows to de ne assistance blocks for BNAR changes in order to present to modelers the way in which the model can be change based on the BNAR changes done in the metamodel.

6. Gruschko et al. [13] propose to coevolve the model using a set of automatic transformations defined previously solving the problems in one of the next three categories: addition, delete or rename. When a BNAR change is found, the user should specify the way that the elements are going to change.

4. Solution strategy

This proposal consists of a strategy where the domain experts that compose metamodels specify the unique solution for BNAR changes in one specific model that conforms to metamodel to be composed. Then, metamodelers write the composition and adaptation in one script in which he/she specifies the original metamodel, additional metamodels for composition, and the model to be adapted. The script must be written in one DSL created for this proposal.

The proposal achieves the metamodels composition and models adaptation based on independent migration transformations, where each one of them is related with one composition instruction applied on the metamodel. Each composition instruction can be based on the several metamodels. Each migration transformation affects just the instances related with the element changed in the metamodel. As a result, each independent migration transformation generates an intermediate metamodel and conformant intermediate model. Figure 2 shows the composition and adaptation strategy.

In this approach, the metamodel MM is composed to the metamodel MM' through that is a set of transformations $_i$ that create intermediate composite metamodels MM_i. Each metamodel MM_i is the result of the composition based on supporting metamodels MM_{sup-1}, MM_{sup-2}, . . . , MM_{sup-n}. In addition, each supporting metamodel changes when the transformation $_i$ is applied generating intermediate sup-porting metamodels MM_{sup-1i} , MM_{sup-2i} , . . . , MM_{sup-ni} . More-over, the model m that conforms to MM, is migrated to m' that conforms to MM', through the migration transforma-tion M that is a set of migrations $_i$ that creates interme-diate models m_i. Each intermediate model m_i conforms to the intermediate metamodel MM_i. The model migration is perform only in the cases that $_i$ is BNAR change i.e. the composition applied on the metamodel breaks the model conformity.

The metamodels composition and models adaptation algorithm is presented in Algorithm 1.

Algorithm 1

```
for all δ in Δ do
    apply(δ, MM1, MMsup-1i, MMsup-2i, ..., MMsup-nj)
    if δ is BNAR then
        for all γ in Γδ do
            apply(γ, μi)
        end for
    else
        for all γ in Γδ do
            apply(γ)
        end for
    end if
end for
```

5. Proposal languages

This approach resolves the metamodels composition and models adaptation by defining two languages. The first one, called metamodel composition language is used to describe the metamodel composition , and the second one, called model adaptation language is used to de ne how to adapt the conformant model m in order to guarantee conformity with the conformant metamodel in case of those changes are BNAR. To address these cases, the model adaptation language is include d in the corresponding instructions of the metamodel composition language.

5.1 Metamodel Composition Language

This DSL includes an instructions catalog of the possible operations that can be applied over several input metamodels in order to generate a unique output composite metamodel. The structure of the DSL consists in the next three operations: 1) Operation import; this operation allows specifying several input metamodels. 2) Operation export; this operation allows specifying the output composite metamodel. 3) Instructions; each instruction specifies a change in the composite metamodel. The DSL has a set of operations that allow metamodelers to define possible changes over the input metamodels in order to construct the composite metamodel, which are defined in the instruction catalog.

This proposal is completeness from the principle that each instruction has high granularity, which implies that the operation cannot be decomposed into smaller operations [17], to ensure unitary changes on the metamodel in the composition process. As a result, the DSL has a catalog made up of 17 instructions. With these instructions metamodelers can make the necessary changes on classes, attributes and references from the input metamodels. Also, metamodelers can include new classes, attributes and references that are not defined in any input metamodel. This kind of operation allows the metamodeler not only make composition, but make changes on the origin metamodel in order to evolve it. Table 1 presents the instruction catalog created for the meta-models composition language. When any instruction make reference to a class, it is necessary to indicate the name of the input metamodel in which the class is placed. In the case that the instruction does not have the name of the input metamodel, the engine will search the class between the classes created before in the composition process.

With this instructions catalog, the metamodels composition language offers a language that supports a great variety of metamodel composition cases. In order to explain how the operations can be used, the next two metamodels presented in Figure 3 will be used.

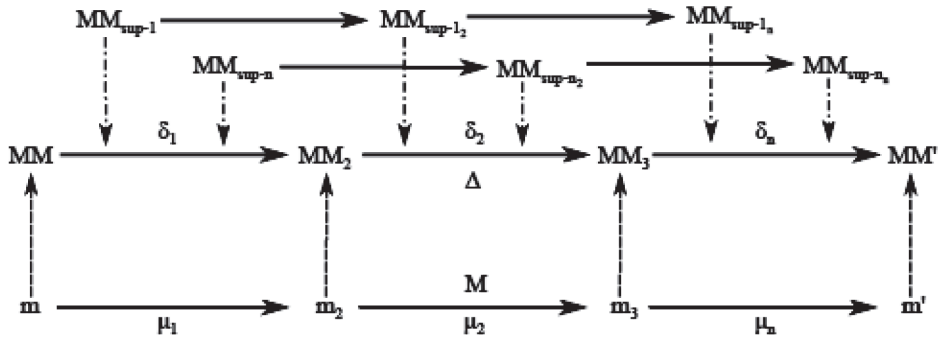
Based on metamodels presented in Figure 3, the script presented in the Listing 1 makes performs a composition process. In this script, lines 1 and 2 imports the metamodels presented in Figure 3; line 3 export the composite metamodel; lines 4 and 6 creates the classes N and M; lines 5, 7, and 8 creates attributes in specified classes; line 9 sets the class V as abstract class; line 10 joins the classes E and N creating a new class named EN; lines 12 and 13 creates new references in the specified classes; and line 13 divides

Table 1. Instruction Catalog for Metamodel Com-position Language

Instruction	Parameters
newClass	Class Name
deleteClass	Class Name
renameClass	Class Name, New ClassName
setAbstractClass	Class Name
setNonAbstractClass	Class Name
joinClasses	New Class Name, Class Name 1, Class Name 2
divideClasses	Class Name, Divided Classes [Divided class name, Divided class attributes, Divided class references]
newAttribute	Class Name, Attribute Name, Type
deleteAttribute	Class Name, Attribute Name
renameAttribute	Class Name, Attribute Name, New Attribute Name
updateAttribute	Class Name, Attribute Name, Type
newReference	Reference Name, Source Class Name, Target Class Name, Containment, Min Cardinality, Max Cardinality
renameReference	Reference Name, New Reference Name
deleteReference	Class Name, Reference Name
updateReference	Class Name, Containment, Min Cardinality, Max Cardinality
newInheritance Reference	Sub Class Name, Super Class Name
deleteInheritance Reference	Sub Class Name

Reference: investigation.

Figure 2. Composition and Adaptation Strategy



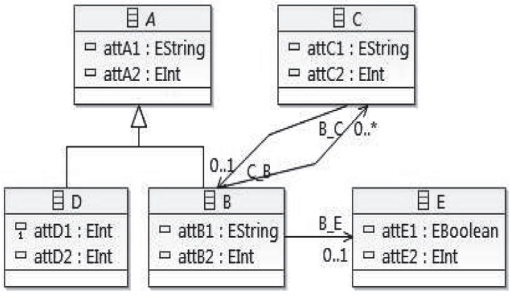
Reference: investigation

the class X creating the classes X1 and X2. As a result the composite metamodel is presented in the Figure 4.

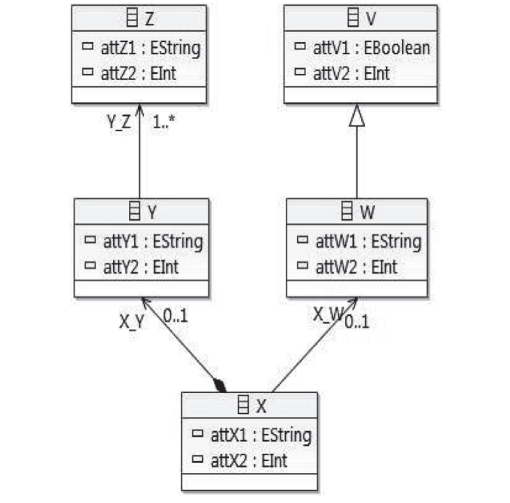
Listing 1: Composition example

```
Listing 1: Composition example
1 import "inputMM1.ecore"
2 import "inputMM2.ecore"
3 export "outputMM"
4 newClass (N)
5 newAttribute (N, attN1, EInt)
6 newClass (M)
7 newAttribute (M.attM1, EInt)
8 newAttribute (B.attB3, EInt)
9 setAbstractClass (inputMM1.V)
10 joinClasses (EN, inputMM1.E, N)
11 newReference (M_Z, M, inputMM2.Z, false, 1, -1)
12 newReference (X_B, exampleMM2.X, exampleMM1.B, ↵
      ↳ false, 0, -1)
13 divideClass (exampleMM2.X [X1, attX1, X_Y, X_B], ↵
      ↳ [X2, attX1, X_Y, X_W])
```

Figure 3. Imported Metamodels Example



a) Metamodel 1



b) Metamodel 2

Reference: investigation

Table 2. Instruction Catalog Clasification

Change Type	Instruction
Non-breaking (NB)	newClass setNonAbstractClass newReference
Breaking and automatically resolvable (BAR)	renameClass renameAtribute renameReference
Breaking and not automatically resolvable (BNAR)	deleteClass setAbstractClass joinClasses divideClasses newAttribute deleteAttribute updateAttribute deleteReference updateReference newInheritanceReference deleteInheritanceReference

Reference: investigation.

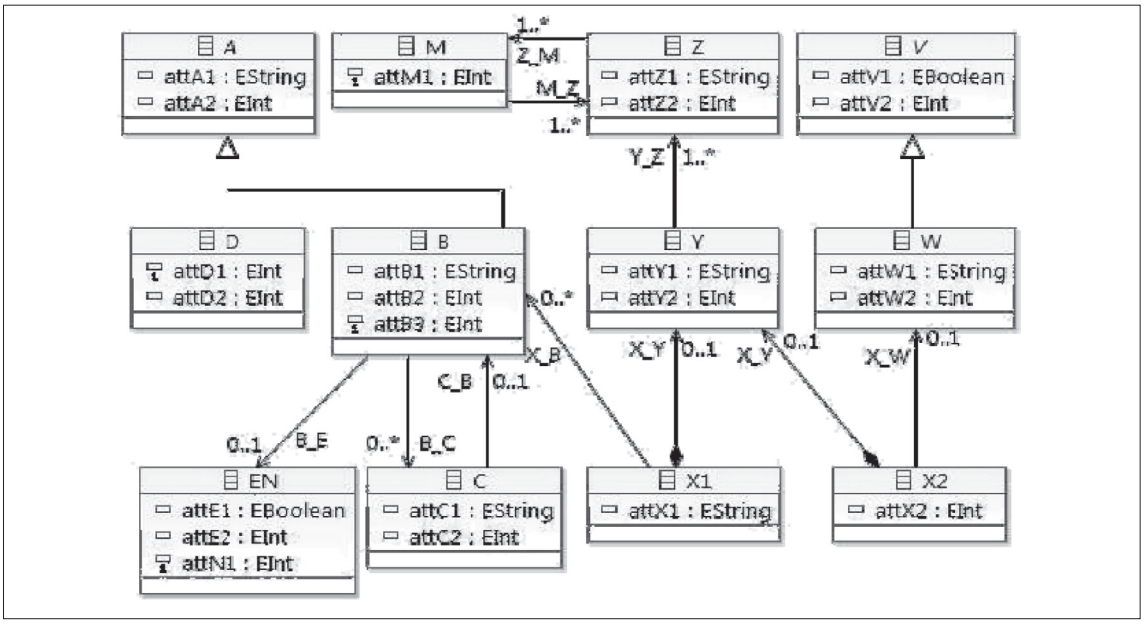
5.2 Model Adaptation Language

The model adaptation language is used to solve the BNAR changes on models that conforms with the composite meta-model. Table 2 presents the classification in NB, BAR, and BNAR instructions of the composition language.

This language consists of a set of instructions that allow metamodelers to specify the changes to be applied to one specific conformant model for the BNAR changes in the metamodel. Due to the granularity level of the metamodel composition language, it is possible to include the related source code of the model adaptation language for each BNAR change creating a block using braces after one BNAR instruction. In addition, the process is sequential, so the order of the instructions guarantee the model's semantics.

The Language grammar is based on Java grammar whose it is possible to allow re-

Figure 4. Composite Metamodel Example

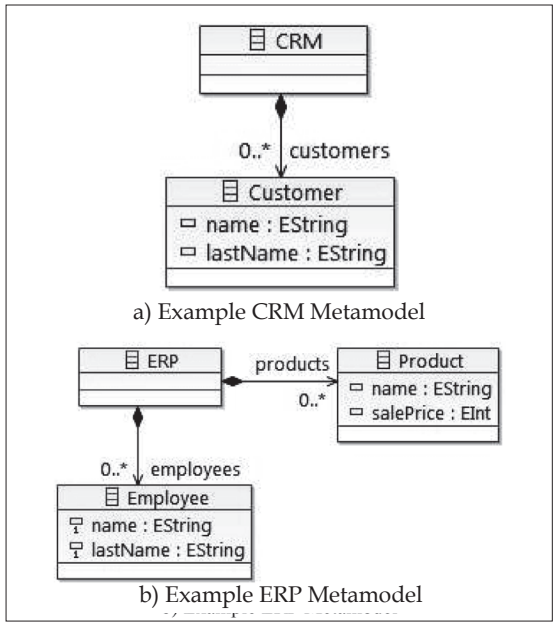


Reference: investigation.

solving: variables declarations, arithmetic operations, compare operations, concatenate operations, logical operations, iteration functions, condition functions, input functions, and output functions. As a result, this language offers the following advantages: 1) the grammar is well known by metamodelers with some experience in Java, and 2) metamodelers can create libraries with reusable scripts to solve adaptation patterns.

In order to explain how the language can be used, the meta-models presented in Figure 7 and the model presented in Figure 6 will be used. Bases on this metamodels, the scrpt presented in Listing 2 is applied. In this script, lines 1 and 2 imports the metamodels presented in Figure 7; line 3 export the composite metamodel; lines 4 and 5 imports the models presented in Figure 6; lines 6, 7, 8, 13, 16, 17, 18, 19 and 20 make changes over the composite metamodel; lines 9, 10, 11, and 12 makes an model adaptation process migrating the instances of the relation customers; and line 14 does not make any change due the class do not have instances with additional information. As a result the composite metamodel and the adapted model are presented in the Figure 4.

Figure 5. Metamodels for Adaptation Example

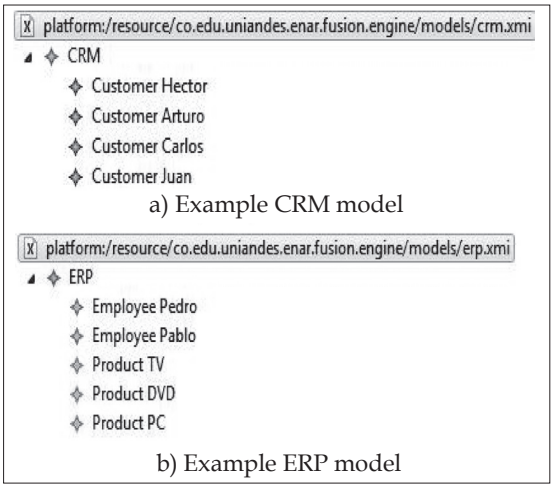


a) Example CRM Metamodel

b) Example ERP Metamodel

Reference: investigation.

Figure 6: Model for Adaptation Example



a) Example CRM model

b) Example ERP model

Listing 2: Adaptation example

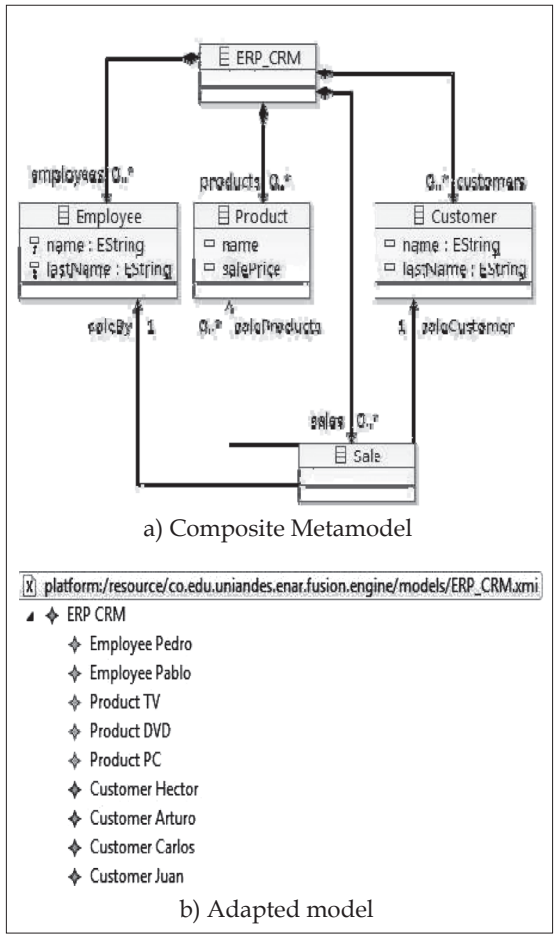
```
1 import "erp.ecore"
2 import "crm.ecore"
3 export "erp_crm"
4 model "crm.xmi"
5 model "erp.xmi"
6 renameClass (erp.ERP, "ERP_CRM")
7 newReference (customers, erp.ERP_CRM, crm.Customer, ↵
      ↳ true, 0, -1)
8 deleteReference (crm.CRM.customers){
9   for(EReference customer: crm.CRM.customers){
10    erp.ERP_CRM.customers.add(customer);
11  }
12 }
13 deleteClass (crm.CRM){
14   System.out.println("There are no instances on the ↵
       ↳ model conform to CRM");
15 }
16 newClass (Sale)
17 newReference (sales, erp.ERP_CRM, Sale, true, 0, -1)
18 newReference (saleBy, Sale, erp.Employee, false, 1, ↵
      ↳ 1)
19 newReference (saleProducts, Sale, erp.Product, ↵
      ↳ false, 0, -1)
20 newReference (saleCustomer, Sale, crm.Customer, ↵
      ↳ false, 1, 1)
```

6. Engine

6.1 Composition Engine

The composition engine of this proposal executes the com-position script sequentially. Once, the engine executes the import operations, it creates in dynamically memory the objects of each metamodel inside the correspondent package. Using the metamodels shown in Figure 3, and the script presented in Listing 1, for each source code line the engine makes the following changes:

Figure 7: Composite metamodel and adapted model.



a) Composite Metamodel

b) Adapted model

Reference: investigation.

1. For lines 1, 2, and 3, the engine loads the element in the correspondent package. The distribution of the elements in dynamic memory is presented in Figure 8a.

2. For lines 4 and 6, the engine creates the classes N and M inside the package outputMM. The distribution of the elements in dynamic memory is presented in Figure 8b.

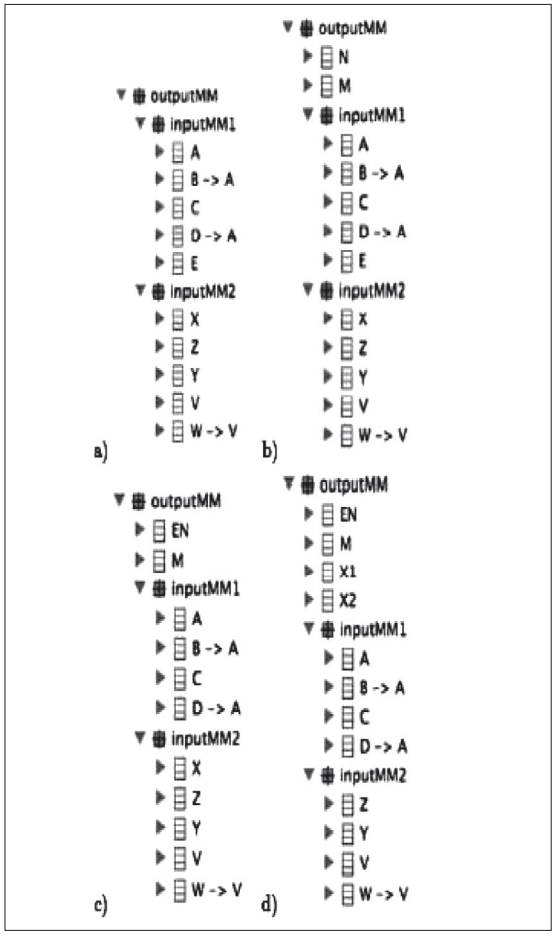
3. For line 10, the class EN is created in the generic pack-age outputMM. However, the classes involved in this operation that are E and N will be deleted from the correspondent packages. The distribution of the elements in dynamic is presented in Figure 8c.

4. For line 13, the classes X1 and X2 are created in the generic package outputMM. However, the class X will be deleted from the correspondent package. The distribution of the elements in dynamic memory is presented in Figure 8d.

Once the composition engine executes the script, the classes from the import metamodels that have not been affected will be translated to the generic package outputMM. Also the packages of the import metamodels will be deleted. As a result, all elements in the composite metamodel will bellow to the generic package.

In the case that the engine finds that one operation can-not be executed the engine will report the mistake and the process will not continue. The reasons in which the process can fail are the follows: 1) the import metamodel does not exist; 2) the class, attribute, or reference required does not exist; 3) in the case of creation of new elements; the class, attribute, or reference related already exist; and 4) after ex-ecuting the script, there are duplicated classes.

Figure 8. Distribution of elements in dynamic memory



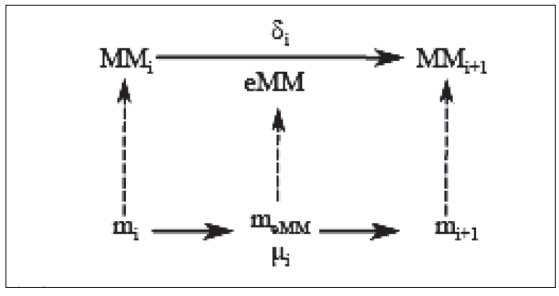
Reference: investigation.

6.2 Adaptation Engine

One metamodel can provide an ontological and a linguistic support for model creation. The ontological support allows metamodels to describe what elements of the reality are represented by model elements, and what are the valid ways to relate them. The linguistic support allows metamodels to de ne the primitives to describe the models, their elements, and their relationships [21]. Then, model elements are onto-logical instances of the types described in the metamodels; but model elements are linguistic instances of the types described in the metamodels [12].

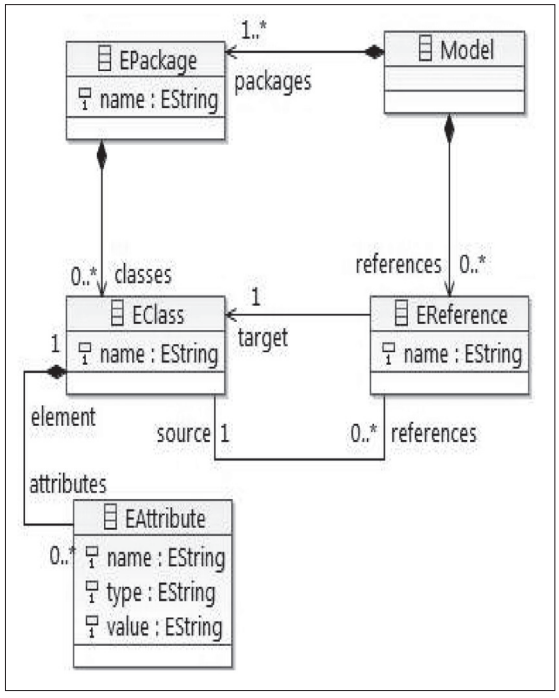
The model adaptation engine makes a dynamic transformation on the model in order to ensure a linguistic conformity with a simplifed version of ECORE metamodel named eMM. Given this transformation, al changes on the model are applied on a temporal model that conform with eMM named M_{eMM}. Figure 9 illustrates the transformation process. In this process, when a BNAR change $_i$ is applied on MM_i due to a composition process, one model transformation dynami-

Figure 9. Temporal Model Transformation



Reference: investigation.

Figure 10. Simplifed ECORE metamodel eMM



Reference: investigation.

cally creates the m_{eMM} that conforms with the eMM. Immediately, the adaptation process $_i$ is performed on the m_{eMM} ensuring linguistic conformity. Finally, an-other transformation dynamically generates m_{i+1} than con-forms with M_{i+1}. Figure 10 show the simpli ed ECORE metamodel eMM.

7. Conclusion

A metamodel composition process, where metamodelers can adapt concepts abstracted in several existing metamodels, is possible. In this approach one DSL allows metamodelers de-ne metamodel composition process. One advantage of this approach is that metamodelers cannot perform illogical com-position operations. Another advantage of this approach is based on the execution of the composition as a set of atomic operations over the input metamodels, each transition uses the modifications done in the previous operations. One more advantage is the creation of metamodels reducing the e ort for metamodelers by getting the elements abstracted in existing metamodels.

The presented approach is simple, completeness and has high granularity, for each composition operation can be done independently and all of them cannot be decompose in smaller operations; as a result, the proposal is adequate to be used by metamodelers in order to create new abstractions through a metamodel based on existing metamodels.

In addition, models adaptation allows transforming models avoiding conformity break out for BNAR changes done in the metamodel due to a composition process. In this approach, one DSL that allows to specify the way in which the model can be transformed in order to regain the conformity with the composite metamodel.

8. References

[1] S. Becker, B. Gruschko, T. Goldschmidt, and H. Koziolek. A process model and classi cation scheme for semiautomatic metamodel evolution. In 1st Workshop MDD, SOA und IT-Management (MSI), GI, GiTO-Verlag, pages 35-46, 2007.

[2] J. Bezivin. On the uni cation power of models. *Software and Systems Modeling*, vol. 4, num. 2: 171-188, 2005.

[3] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating coevolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference*, 2008. EDOC'08. 12th International IEEE, pages 222/231. IEEE, 2008.

[4] Eclipse Foundation. Eclipse Modeling Framework Project Dec 16th, 2008 - *EMF*: *Eclipse Modeling Framework*, 2nd Edition. (EMF).

[5] M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In OOPSLA. 6th Workshop on Domain Speci c Modeling, pages 123/139, 2006.

[6] H. Flórez. Domain Speci c Language for Metamodel Composition. In The

2012 International Conference on Software Engineering Research and Practice (SERP'12), 2012.

[7] H. Flórez. Model Transformation Chains as Strategy for Software Development Projects. In The 3rd International Multi-Conference on Complexity, Informatics and Cybernetics (IMCIC 2012), Orlando, 2012.

[8] H. Flórez, M. Sanchez, J. Villalobos, and G. Vega. Coevolution Assistance for Enterprise Architecture Models. In Models And Evolution (ME 2012) Workshop at The ACM/IEEE 15th International Conference on Model Driven Engineering Languages And Systems (MoDELS 2012), Innsbruck, 2012.

[9] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh. Providing support for model composition in metamodels. In Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, page 253. IEEE, 2007.

[10] K. Garces, F. Jouault, P. Cointe, and J. Bezivin. Adaptation of Models to Evolving Metamodels. Research Report RR-6723, INRIA, 2008.

[11] K. Garces, F. Jouault, P. Cointe, and J. Bezivin. Managing Model Adaptation by Precise Detection of Metamodel Changes. In Proceedings of the 5th European Conference on Model Driven Architecture-Foundations and Applications, pages 34{49. Springer-Verlag, 2009.

[12] P. Gomez, M. Sanchez, H. Florez, and J. Villalobos. Co-Creation of Models and Metamodels for Enterprise Architecture Projects. In Extreme Modeling (XM 2012) Workshop at ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2012), Innsbruck, 2012.

[13] B. Gruschko, D. Kolovos, and R. Paige. Towards synchronizing models with evolving metamodels. In Proceedings of the International Workshop on Model-Driven Software Evolution, 2007.

[14] K. Hassam, S. Sadou, V. Gloahec, and R. Fleurquin. Assistance System for OCL Constraints Adaptation During Metamodel Evolution. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on, pages 151-160. IEEE, 2011.

S. J. Henriksson, F. Heidenreich, J. Johannes, S. Zschaler, and U. A mann. Extending grammars and metamodels for reuse: the Reuseware approach. Software, IET, vol. 2, num. 3: 165-184, 2008.

[15] M. Herrmannsdoerfer, S. Benz, and E. Juergens. COPE-automating coupled evolution of metamodels and models. ECOOP 2009{Object-Oriented Programming, pages 52-76, 2009.

[16] M. Herrmannsdoerfer, D. Ratiu, and G. Wachsmuth. Language evolution in practice: The history of GMF. Software Language Engineering, pages 3-22, 2010.

[17] M. Herrmannsdoerfer, S. Vermolen, and G. Wachsmuth. An extensive catalog of operators for the coupled evolution of metamodels and models. Software Language Engineering, pages 163-182, 2011.

[19] G. Karsai, M. Maroti, A. Ledeczi, J. Gray, and J. Sztipanovits. Composition and cloning in modeling and meta-modeling. Control Systems Technology, IEEE Transactions on, 12(2):263{278, 2004.

[20] D. Kolovos, R. Paige, and F. Polack. The epsilon object language (eol). In Model Driven Architecture{Foundations and Applications, pages 128{142. Springer, 2006.

[21] T. Kuhne. Matters of (meta-) modeling. Software and Systems Modeling, vol. 5, num. 4: 369-385, 2006.

[22] A. Ledeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti. On metamodel composition. In Control Applications, 2001. (CCA'01). Proceedings of

the 2001 IEEE International Conference on, pages 756-760. IEEE, 2001.

[23] B. Meyers, M. Wimmer, A. Cicchetti, and J. Sprinkle. A generic in-place transformation-based approach to structured model co-evolution. In 4th Int. Workshop on Multi-Paradigm Modeling, 2010.

[24] P.-A. Muller, F. Fondement, and B. Baudry. Modeling Modeling. In ACM/IEEE 12th International Conference on Model Driven Engineering Languages & Systems MODELS 2009, pages 2-16, 2009.

[25] A. Occello, A. Dery-Pinna, M. Riveill, and G. Kniesel. Managing Model Evolution Using the CCBM Approach. In Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the, pages 453-462. IEEE, 2008.

[26] J. Oldevik. Transformation composition modelling framework. In Distributed Applications and Interoperable Systems, pages 1135-1136. Springer, 2005.

[27] L. Rose, D. Kolovos, R. Paige, and F. Polack. Enhanced automation for managing model and metamodel inconsistency. In Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on, pages 545-549. IEEE, 2009.

[28] L. Rose, D. Kolovos, R. Paige, and F. Polack. Model migration with epsilon ock. Theory and Practice of Model Transformations, pages 184-198, 2010.

[29] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. EMF: eclipse modeling framework. Addison-Wesley Professional, 2008.