Hybrid Approach for Metamodel and Model Co-evolution

Fouzia Anguel, Abdelkrim Amirat, Nora Bounour

Hybrid approach for metamodel and model co-evolution

Fouzia Anguel[1,3], Abdelkrim Amirat[2] and Nora Bounour[3]

[1] Chadli Bendjedid University, El Tarf, Algeria.
fanguel@yahoo.fr
[2] LiM Laboratory, Mohammed Chérif Messaadia University, Souk-Ahras, Algeria
abdelkrim.amirat@yahoo.com
[1,3] LISCO Laboratory. Badji Mokhtar University.Annaba, Algeria
nora_bounour@yahoo.fr

Abstract . Evolution is an inevitable aspect which affects metamodels. When metamodels evolve, model conformity may be broken. Model co-evolution is critical in model driven engineering to automatically adapt models to the newer versions of their metamodels. In this paper we discuss what can be done to transfer models between versions of a metamodel. For this purpose we introduce hybrid approach for model and metamodel co-evolution, that first uses matching between two metamodels to discover changes and then applied evolution operators to migrate models. In this proposal, migration of models is done automatically; except, for non resolvable changes, where assistance is proposed to the users in order to co-evolve their models to regain conformity.

Keywords: Metamodel evolution, Model migration, co-evolution, matching, evolution operator.

1 Introduction

In Model-Driven Engineering (MDE)[1], metamodels and domain-specific languages are key artifacts as they are used to define syntax and semantics of domain models [1]. Since in MDE metamodels are not created once and never changed again, but are in continuous evolution, different versions of the same metamodel are created and must be managed [2]. The evolution of metamodels is a considerable challenge of modern software development as changes may require the migration of their instances. Works in this direction exist already. Several manual and semi-automatic approaches for realizing model migration have been proposed. Each approach aims to reduce the effort required to perform this process. Unfortunately, in several cases it is not possible to automatically modify the models to make them conform to the updated metamodels. This is so because certain changes over metamodels require introducing additional information into the conformant model. In the literature, three general approaches to the migration of models exist: manual, state-based, operator- based [3]. Manual approaches are tedious and error prone. State-based approaches also called difference-based approaches allow synthesizing a model migration based on the dif-

ference between two metamodel versions. In contrast, operator-based approaches allow to incrementally transforming the metamodel by means of coupled operations which also encapsulate the corresponding model migration. They allow capturing the intended model migration already when adapting the metamodel. A major drawback of the later approach has been overly tight coupling between the tool performing the migration, and the recorder tracking the changes made to the models.

Usually, existing approaches try to find how to best accomplish model co-evolution. Essentially, we can define two main requirements: the correctness of migration and minimizing the effort of migration by automating as far as possible the process.

In this paper, we propose an alternative solution to model migration which combines state-based and operator based principles to co-evolve models and metamodels. Our vision to resolve this problem is to generate evolution strategies with their corresponding model migration strategies. We focus on including users decisions during metamodel and model co-evolution process to ensure semantic correctness of evolved models.

The rest of the paper is structured as follows. Section 2 gives an overview of basic concepts and describes the metamodel and model co-evolution problem Section 3 presents our proposed approach for solving the model co-evolution problem. In section 4, we present some proposed approaches in the past and situates our solution. Section 5 presents some guidelines to implement proposed framework. Finally, section 6 concludes and gives some future works.

2 Background

2.1 Models and metamodels

In this section we present the central MDE definitions used in this paper. The basic assumption in MDE is to consider models as first-class entities. An MDE system basically consists of metamodels, models, and transformations. A model represents a view of a system and is defined in the language of its metamodel [1]. In other words, a model contains elements conforming to concepts and relationships expressed in its metamodel [4]. A metamodel can be given to define correct models. In the same way a model is described by a metamodel, a metamodel in turn has to be specified in a rigorous manner; this is done by means of meta-metamodels [5]. This may be seen as a minimal definition in support of the basic MDE principle “Everything is considered as a model” [1]. The two core relations associated to this principle are called representation “Represented by” and conformance “Conform To”. A model conforms to a metamodel, when the metamodel specifies every concept used in the model definition, and the models uses the metamodel concepts according to the rules specified by the metamodel [1].

In this respect, the object management group (OMG) [6] has introduced the four level architecture which organizes artifacts in a hierarchy of model layers (M0, M1, M2, and M3). Models at every level conform to a model belonging to the upper level. M0 is not part of the modeling world as depicted in Fig.1, so the four level architec-

ture should more precisely be named (3+1) architecture [1]. One of the best known metamodels in the MDE is the UML (Unified Modeling Language) metamodel; MOF (Meta-Object Facility) is the metametamodel of OMG that supports rigorous definition of modeling languages as UML [6].

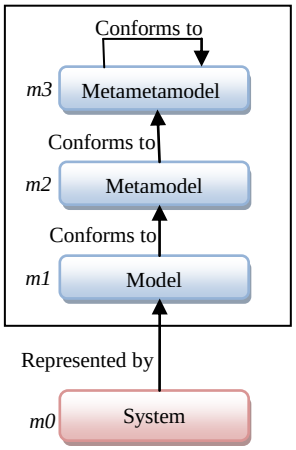

Fig. 1. The 3+1 MDA organisation [2]

2.2 Metamodel evolution and model co-evolution

Metamodels may evolve in different ways, due to several reasons [2]: during design, alternative metamodel versions are developed and well-known solutions are customized for new applications. During implementation, metamodels are adapted to a concrete metamodel formalism supported by a tool. During maintenance, errors in a metamodel are corrected. Furthermore, parts of the metamodel are redesigned due to a better understanding or to facilitate reuse. The addition of new features and/or the resolution of bugs may change metamodels, thus causing possible problems of inconsistency to existing models which conform to the old version of the metamodel and may become not conform to the new version. Therefore to maintain consistency, metamodel evolution requires model adaptation, i.e., model migration; so these two steps are referred as model and metamodel co-evolution [7]. Metamodel and model co-evolution is a term that denotes a coupled evolution of metamodels and models [7], which consists to adapt (co-evolve) the models conforming to the initial version of the metamodel, such that they conform to the target (evolved) version, preserving the intended meaning of the initial model if possible [7], as illustrated in Fig.2. Furthermore, model adaptations should be done by means of model transformations [8]. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to the evolved version of the given metamodel [4].

A number of works proposed the classification of metamodel changes according to their corrupting effects. Metamodel changes are grouped on three categories [9]:

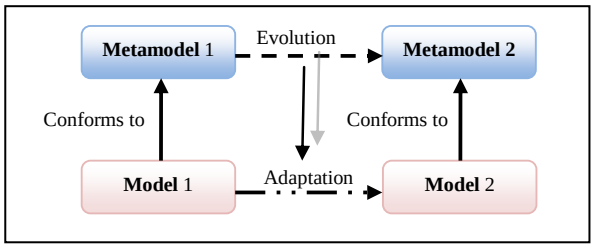

Fig. 2. Model Co-evolution [7]

- Not breaking changes, changes occurring in the metamodel don't break the models conformance to the metamodel.
- Breaking and resolvable changes, changes occurring in the metamodel do break the models, which can be automatically resolved.
- Breaking and non-resolvable changes, changes do break the models and cannot be automatically resolved and user intervention is required.

However, a uniform formalization of metamodel evolution is still lacking. The relation between metamodel and model changes should be formalized in order to allow reasoning about the correctness of migration definitions.

2.3 Logic programming

Logic programming is a programming paradigm based on formal logic [10]. A program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain. Major logic programming language families include Prolog, Answer Set Programming (ASP) and Datalog. In all of these languages, rules are written in the form of clauses (H :- B1, …, Bn). These clauses are called definite clauses or Horn clauses and are read declaratively as logical implications (H if B1 and … and Bn). Logic programming is used in artificial Intelligence knowledge representation and reasoning.

We have find this formalism very powerful to represent relationships between changes and consequently, from an initial set of changes inferring all possible evolution strategies. Currently, to our best knowledge, there is no approach that uses an intelligent reasoning for defining model migrations. Therefore, we have integrated logic programming in our proposal to resolve model co-evolution problem.

3 Proposed Approach

In this section we describe our proposal to ensure the co-evolution of model with their metamodels. The overall evolution and co-evolution process is presented in Fig.3.

Our approach is hybrid because it exports techniques from state-based and operator based approaches and uses also a reasoning mechanism from artificial intelligence. It contains four phases: changes detection, generation and validation of evolution strategies, determination of migration strategies and migration of models.

In the first step; differences between two metamodel versions need to be determined by using matching technique. In the second step we use an inference engine to generate different evolution strategies by assembling atomic changes in possible compound ones; in the third step we explore a library of operators to obtain different migration procedures, which will be assembled to constitute migration strategies. In the last step users employ a selected evolution strategy and consequently, the migration strategy will be applied over a specific model conforming to the old version in order to obtain a new model conforming to the newer metamodel version.

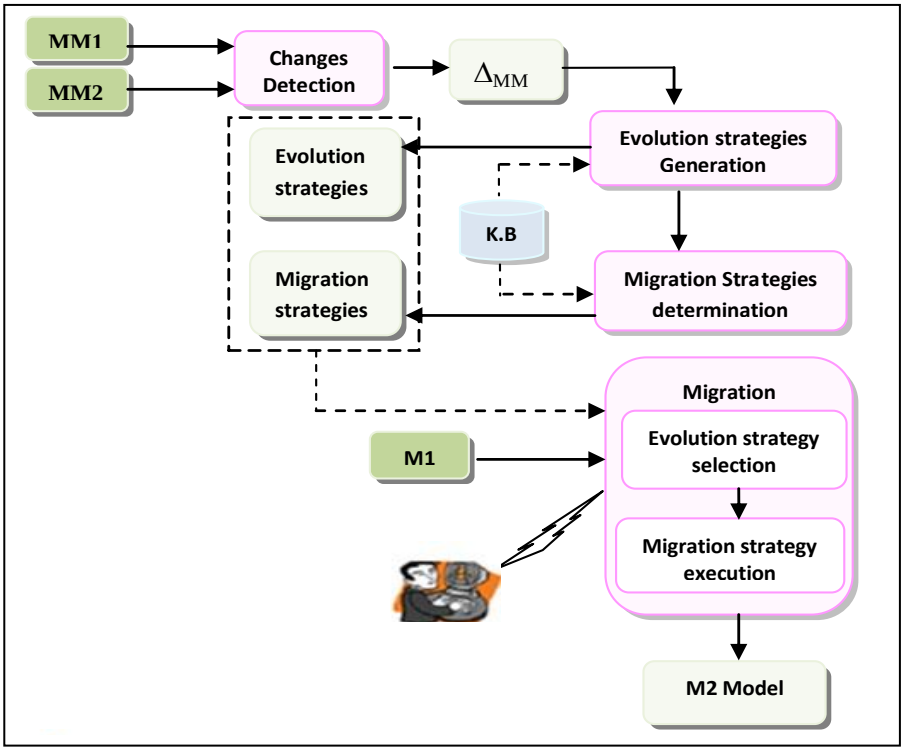

Fig. 3. An overview of metamodel and model co-evolution process

3.1 Detection of changes

The detection of differences between models is essential to model development and management practices. Thus evolution from one metamodel version to the next can be described by a sequence of changes. Understanding how metamodels evolve or dis-

covering changes that have been performed on a metamodel is a key requirement before undertaking any migration operation on models to co-evolve them. In fact, we distinguish two ways for discovering changes: matching approaches and recording approaches [11]. In Our approach, for detecting the set of changes performed to the older version of the metamodel in order to produce the new one, we use generic algorithm. Whereas current generic approaches only support detecting atomic changes, some language-specific approaches also allow detecting composite changes; but only for one specific modeling language. Primitive differences between metamodels versions are classified in three basic categories: additions, deletions, and updates of metamodel elements. These differences represent elementary changes (i.e. atomic).
In fact, composite or compound changes have been already considered in previous works like [12,13]. But, we envision tackle the problem differently. We call evolution strategy a possible sequence of changes; here changes are either elementary or composite. Thus, a set of composite changes is inferred from the detected set of atomic changes, by using rules that define composite changes in terms of atomic changes. This mechanism is detailed in the following section.

3.2 Generation and validation of evolution strategies

Detected differences are represented as elementary changes specifying fine-grained changes that can be performed in the course of metamodel evolution. There are a number of primitive metamodel changes like create element, rename element, delete element, and so on. One or more of such primitive changes compose a specific metamodel adaptation However, this granularity of metamodel evolution changes is not always appropriate.

Often, intent of the changes may be expressed on a higher level. Thus, a set of atomic changes can have together the intent of a composite change. For example, generation of a common superclass sc of two classes c1 and c2 can be done through successive applications of a list of elementary changes, such as 'Add_class sc', 'Add_reference from c1 to sc', and 'Add_reference' from c2 to sc. One way to resolve the problem of identifying composite changes is to use operation recording. But, this solution has some drawbacks.

In our proposal, we use logical predicate with the language Prolog. Horn clauses are used to represent knowledge. Therefore, we formally characterize changes. Detected atomic changes are represented as positive clauses (i.e. facts) and composite changes are specified by rules such as Left hand side contains the composite change and the Right hand side contains a set of associated atomic changes. Thus, the applicability of a compound change can be restricted by conditions in the form of rules. According to this principle, we have formalized a knowledge base. The definition of changes is inspired from the literature [7, 14]. The knowledge base is used by the inference engine to generate possible evolution strategies. Finally, evolution strategies must be validated. This step consists of applying each evolution scenario defined by the strategy on the old input version of the metamodel. If it results the newer input version then the tested strategy is valid and it is retained else the strategy in test is rejected. The final output is a set of valid evolution strategies.

C : set of classes
A : set of attributes
Auxiliary predicates
subclasse(s : C, c:C) : s is subclass of c
added_class(c:C) : c is added to the metamodel
added_attribute(a:A,c:C) : a is added to the class c of the metamodel.
deleted_attribute(a:A,c:C) : a is deleted from the class c of the metamodel.
Is_attribute_of(a:A,c:C) : a is an attribute of the class c.
added_supertype ((s : C, c:C) . specialization/generalization reference is added between s and c.
deleted _supertype ((s : C, c:C) : specialization/generalization reference between s and c is removed.
added-reference (r :R, s: C, d:C) : r is an added reference having as source s class and d as target class.
Extract-superclass(sc ,c1,c2 :C) :- added_class(sc: C),
added_supertype ((sc : C, c1:C),
added_supertype ((sc : C, c2:C).

Complex changes are specified through rules. As an instance, we consider extract super class operation where a class is generalized in a hierarchy by adding a new general class and two references to their subclasses.

3.3 Determination of migration strategies

In this step we import techniques of operator based-approaches. We use in this phase a library of operators. Thus, we specify a change as an evolution operation. An operation evolution can be either simple or composite and every operation is defined through a set of parameters. We associate to it information about how to migrate corresponding models in response to a metamodel evolution forming a migration procedure. Migration procedure is encoded as a model transformation that transforms a model such that the new model conforms the metamodel undergoing the change. Furthermore, we explicitly specify in migration procedures some assistance specifications for each change requiring additional information from user to solve it. This makes our library different of that used in previous works [13]. The library does not contain evolution steps but only the migration procedure referenced with evolution operation. In our proposal we take from the library migration procedures corresponding to changes in the evolution strategy; after their instantiation, we assemble them to constitute the complete migration strategy which will be associated to the evolution strategy. The final result in this step is a set of couples (evolution strategy, migration strategy) specified to co-evolve input models conforming to the specified metamodel.

3.4 Migration

This phase takes as input an instance model conforming to the initial metamodel. This model is also called user model. To transform the model to newer version of the met-

amodel, firstly one of available evolution strategies previously inferred is considered. According to the taken evolution strategy associated migration strategy will be automatically generated and then applied to the input model. For breaking and irresolvable changes, the system assists establish adequate migration procedure by presenting alternative solutions. Additionally, users can provide additional information to complete the change on the model if necessary. For instance, if the new attribute must be initialized, the user must also be requested for the initial value. If the user is satisfied by the resulted model the process is achieved, otherwise he can try again by selecting other proposed evolution strategy and the process continues so that, until user satisfaction or no choice is available.

3.5 Implementation

In this section, we give details and technical choices made to implement a prototype of the proposed framework. As meta-metamodel, we use Ecore from the Eclipse Modeling Framework (EMF) [16]. However, our approach is not restricted to Ecore, as it can be transferred to all object-oriented metamodeling formalisms.

For the definition of rules specifying knowledge base used to infer evolution strategies, we have adopted an adequate formalism for logic programming Prolog [10]. Prolog is chosen because in one hand it is a language of knowledge representation [17] and in the other hand using inference rules eliminates programming to get eventual compound changes, the task is performed by the inference engine of Prolog. Furthermore, Prolog interpreters are developed in several languages, which facilitates the use of the prolog formalism.

The computation of the differences between metamodel versions is performed with The Eclipse plug-in EMF Compare [18]. This tool provides algorithms to calculate the delta between two versions of a model and visualizes them using tree representations. EMF Compare is capable of detecting the following types of atomic operations:

- Add: A model element only exists in the revised version.
- Delete: A model element only exists in the origin version.
- Update: A feature of a model element has a different value in the revised version than in the origin version.
- Move: A model element has a different container in the revised version than in the origin version.

4 Related Works

In this section we will give an overview of current metamodel and model co-evolution approaches and already implemented systems. Over the last few years, the problem of metamodel evolution and model co-evolution has been investigated by several works like [4], [7-9], [12-13], [19-21]. Currently, there are several approaches that focus on resolving inconsistencies occurring in models after metamodel evolution [3], a classification of these model migration approaches is proposed in [3]. This classification

highlights three ways to identify needed model updates: manually, based on operators, and by using metamodel matching. When manually approaches like in [19-21], updates are defined by hand. In operators based approaches, like [7],[13], metamodels changes are defined in terms of co-evolutionary operators [14]. Those operators define conjointly the evolution on the metamodel and its repercussion on the models. Finally, in metamodel matching, like [4], [9], [12], versions of metamodels are compared and differences between them are used to semi-automatically infer a transformation that expresses models updates. Manual specification approach like Flock [21] is very expressive, concise, and correctness is also assured but finds difficulties with large metamodels since there is no tool support for analyzing the changes between original and evolved metamodels [22]. Operator based approaches like [13] ensure expressiveness, automaticity, and reuse [23], it was been perceived as strong in correctness, conciseness and understandability [22] but its lack is in determining which sequence of operations will produce a correct migration. Analysis of existing model co-evolution approaches, and comparison results of some works [3], [24-25] has yielded guidance for defining some requirements to our approach. To take advantage of state-based and operator-based approaches, previously discussed. We have proposed an alternative solution where we applied a hybrid approach to define model migration. The solution presented in this paper has a number of similarities with the techniques illustrated in [13], but it differs from this approach because it takes as input results of a matching process. Therefore, it permits evolving models with different tools. Another, strength of our solution is the proposed reasoning mechanism, which allows finding different evolution strategies and consequently different migration strategies. Proposed solution minimizes as far as possible the user effort to migrate models. Thus user intervention is limited to a control task in the end of the process to validate results which permits to increase expressivity and correctness.

5 Conclusion

In this paper we have proposed an alternative solution to automate the co-evolution of models and metamodels. In our proposal we use a hybrid approach. It takes advantages from state-based and operator based approaches. This solution consists of using a library of coupled operation and also a knowledge base of changes definition

The benefits of this approach are numerous, notably automaticity of the co-evolution is augmented compared with other techniques because even for changes requiring specific information, we have predict automatic model migration with user assistance. Moreover, our solution is independent from any modeling environment. It is easily adapted to various modeling environment. Using an intelligent logic mechanism to infer compound changes and evolution strategies increase effectiveness of our proposal. This makes our solution distinguishable from existing works.

However, currently the evaluation of the proposed framework is not performed. For a complete validation, we will conduct case studies with industrial models. In the long term, we want to study the possibilities to extend our solution to support repre-

sentation of semantic in models and preserving semantics within the migration process as introduced in [26].

References

1. Bézivin, J.:On the Unification Power of Models. In: Software and systems Modeling (SoSyM.), vol. 4(2), pp. 171–188(2005)
2. Favre, J.M.:Meta-model and model co-evolution within the 3D software space. In: International Workshop on Evolution of Large-scale Industrial Software Applications ELISA'03, Amsterdam. pp 98–109 (2003)
3. Rose, L.M., Kolovos, D.S., Paige, R.F., Polack F.A.C., :An analysis of approaches to model migration. In: Joint MoDSE-MCCM Workshop (2009)
4. Garcès, K., Jouault F., Cointe P., Bézivin J.:Managing Model Adaptation by Precise Detection of Metamodel Changes. In: ECMDA-FA'09. LNCS Springer, vol. 5562, , pp 34-49 (2009)
5. Amirat, A., : Contribution à l'élaboration d'architectures logicielles à hiérarchies multiples, Thèse de Doctorat en Informatique, Université de Nantes, France (2010)
6. OMG: MOF QVT Final Adopted Specification. http://www.omg.org/docs/ptc/05-11-01.pdf, (2005)
   Wachsmuth, G.: Metamodel adaptation and model co-adaptation. In: ECOOP'07.LNCS Springer, vol. 4609, pp. 600-624 (2007)
7. Amirat, A., Menasria, A., ne Gasmallah :Evolution Framework for Software Architecture using Graph Transformation Approach. In: The 12th International Arab Conference on Information Technology (ACIT'2011), December 11-14, Riyadh, Saudi Arabia, pp. 75-82, (2011).
8. Gruschko, B., Kolovos, D.S., Paige, R.F.:Towards synchronizing models with evolving metamodels. In: International Workshop on Model-Driven Software Evolution (2007)
9. Savoy, J.: Introduction à la programmation logique Prolog , available http://members.unine.ch/jacques.savoy/lectures/SemCL/Prolog.pdf (2006)
10. Didonet Del Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.:AMW: A Generic Model Weaver. In: IDM'05 Premières Journées sur l'Ingénierie Dirigée par les Modèles, Paris (2005)
11. Cicchetti, A.:Difference Representation and Conflict Management in Model-Driven Engineering, Phd thesis (2008)
12. Herrmannsdoerfer, M., Benz, S., Juergens, E.:Automatability of Coupled Evolution of Metamodels and Models in Practice. In: MoDELS'08. LNCS Springer, vol. 5301, pp. 645-659 (2008)
13. Herrmannsdoerfer, M., Vermolen, S. D., Wachsmuth, G.:An Extensive Catalog of Operators for the Coupled Evolution of Metamodels and Models. In: SLE'10, LNCS, Springer, Berlin, vol. 6563,pp. 163–182 (2011)
14. Jouault, F., Kurtev, I.:Transforming models with ATL. In:Model Transformations in Practice Workshop at MoDELS Montego Bay, Jamaica, pp. 128–138 (2005)
15. EMF Eclipse Modeling Framework. http://www.eclipse.org/emf
16. Gaizauskas, R.and Humphreys , K. : XI A Simple Prolog-based Language for Cross-Classification and Inheritance. In Proceedings of the 7th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA96), Sozopol, Bulgaria, pp. 86-95, (1996)
17. EMFCompare, Eclipse modeling Project, http://www.eclipse.org/emf/compare/

18. Sprinkle, J., Karsai, G.:A domain-specific visual language for domain model evolution. Journal of Visual Languages and Computing, vol. 15, pp. 291-307 (2004)
19. Narayanan, A., Levendovszky, T., Balasubramanian, D., Karsai, G.:Automatic domain model migration to manage metamodel evolution. In: MODELS'09, LNCS Springer, vol. 5795, pp. 706-711 (2009)
20. Rose, L.M., Kolovos, D.S., Paige, R.F.,Polack, F.A.C.: Model migration with Epsilon Flock. In: ICMT'10 LNCS Springer,vol. 6142, pp. 184-198 (2010)
21. Rose, L.M., Herrmannsdoerfer, M., Mazanek, S., Gorp, P.V., Buchwald, S., Horn, T., Kalnina, E., Koch, A., Lano, K., Schätz,B., Wimmer, M.:Graph and model transformation tools for model migration. In: Software and System Modelling journal (2012).
22. Herrmannsdoerfer, M.: COPE – A Workbench for the coupled evolution of metamodels and models. In. SLE'10, pp. 286-295 (2010)
23. Iovino, L., Pierantonio, A., Malavolta, I.:On the Impact Significance of Metamodel Evolution in MDE. In: Journal of Object Technology, vol. 11, no. 3, pp. 1–33 (2012)
24. M.Herrmannsdörfer, G. Wachsmuth: "Coupled Evolution of Software Metamodels and Models". Book chapter pp 33-63. In "Evolving Software Systems". Mens, Tom, Serebrenik Alexander, Cleve, Anthony (Eds.), 404 p, Springer (2014)
25. Cicchetti, A., Ciccozzi, F.:Towards a Novel Model Versioning Approach based on the Separation between Linguistic and Ontological Aspects", in ME 2013 Models and Evolution Workshop, pp 58-65 (2013)