Towards a Novel Model Versioning Approach based on the Separation between Linguistic and Ontological Aspects

Antonio Cicchetti and Federico Ciccozzi

School of Innovation, Design and Engineering
Mälardalen University, SE-721 23, Västerås, Sweden
antonio.cicchetti@mdh.se, federico.ciccozzi@mdh.se

Abstract. With the increasing adoption of Model-Driven Engineering (MDE) the support of distributed development and hence model versioning has become a necessity. MDE research investigations targeting (meta-)model versioning, conflict management, and model co-evolution have progressively recognized the importance of tackling the problem at higher abstraction level and a number of solving techniques have been proposed. However, in general existing mechanisms *hit the wall* of semantics, i.e. when not only syntax is involved in the manipulations the chances for providing precision and automation are remarkably reduced.
In this paper we illustrate a novel version management proposal that leverages on the separation between linguistic and ontological aspects involved in a (meta-)modelling activity. In particular, we revisit the main versioning tasks in terms of the mentioned separation. The aim is to maximize the amount of versioning problems that can be automatically addressed while leaving the ones intertwined with domain-specific semantics to be solved separately, possibly by means of semi-automatic techniques and additional precision.

1 Introduction

Model-Driven Engineering (MDE) promises to reduce software development complexity by shifting the focus from coding to modelling. Models become first-class citizens and they represent abstractions of real-phenomena tailored to a specific purpose. In this respect they are an appropriate composition of concepts, whose well-formedness is specified by means of a metamodel. Moreover, model transformations are exploited to manipulate models to perform analysis and generate code. Given the relevance gained by models, they are expected to be affected by the same evolutionary pressure source code experienced in the past. Therefore, if MDE approaches are not able to provide evolution support at least comparable with the one existing for text-based software development, MDE adoption would be remarkably hindered.

In the latest years the need for appropriate support of model evolution has been largely recognized and addressed by a number of research works, including differencing, storing versions, managing merges and possible conflicts, supporting metamodel evolution and corresponding model migrations. In particular, model differencing covered both language-specific and agnostic cases, model changes have been tackled both in a state-based and operation-based manner, mechanisms have been introduced to detect divergences between concurrent manipulations of the same model and provide possible reconciliation strategies. Moreover, techniques have been developed to detect metamodel changes, classify them in terms of effects on existing model instances, and provide corresponding migration countermeasures ranging from manual to automatic.

Given the high abstraction level of modelling activities, mixing syntax and semantics is unavoidable; unfortunately, when semantics comes into play, versioning problems become more complex to manage and very often they cannot be dealt with automatically. In other words, automation support has typically to be reduced and user intervention is required to keep the desired degree of precision.

In this paper we propose to enhance automation opportunities by defining a novel methodology for all activities involved in model versioning. The main idea is to exploit the separation between linguistic and ontological aspects of a model [1] and address them separately. In particular, linguistic aspects are those related to the structural correctness of a model, while ontological aspects pertain to the specific domain taken into account (please, see Section 2 for a more precise definition of these aspects). In this way the evolution of the linguistic part, that is expected to be mainly syntactic and hence easier to manage, can be supported by automated mechanisms, whereas the ontological part can be provided with more precise domain-specific versioning support and possibly offer semi-automatic management. Based on the separation mentioned so far, this work revisits the current techniques developed for model version management and outlines a research agenda to cover all the aspects of model versioning.

The structure of the paper is as follows. Section 2 depicts the motivations underneath the proposed methodology as well as the related works in the area. In Section 3 we propose a research agenda to cover the different aspects of model versioning with our novel methodology and we conclude the paper with an outlook in Section 4.

2 Background and Related Work

In MDE, models are commonly defined as abstractions of real phenomena, by means of a given modelling purpose in mind, pursuing a simplification of the reality [2]. In this respect, (meta-)modelling activities carry along not only the syntax by which concepts are expressed (either textual, graphical, or a combination of both), but also the underlying semantics of the application domain taken into account. In general these two aspects are not clearly distinguishable, since part of the semantics can be intertwined with the adopted syntax and structural constraints.

Kühne proposed an alternative separation between those two aspects by introducing *linguistic* and *ontological* matters of (meta-)modelling [1]. In particular, the linguistic can be referred to constraints and rules that define the structural correctness of a model. For instance, a class must have a unique name within a model, or a relationship shall have a source and a target model element. On the contrary, ontological aspects are those pertaining to the domain taken into account, and exploit structural compositions to prescribe domain-specific well-formedness. In other words, they create a new logical abstraction level by specializing (groups of) concepts at lower levels of abstraction. Notably, the class `person` must have a name, a surname, and an age greater than 0 to be

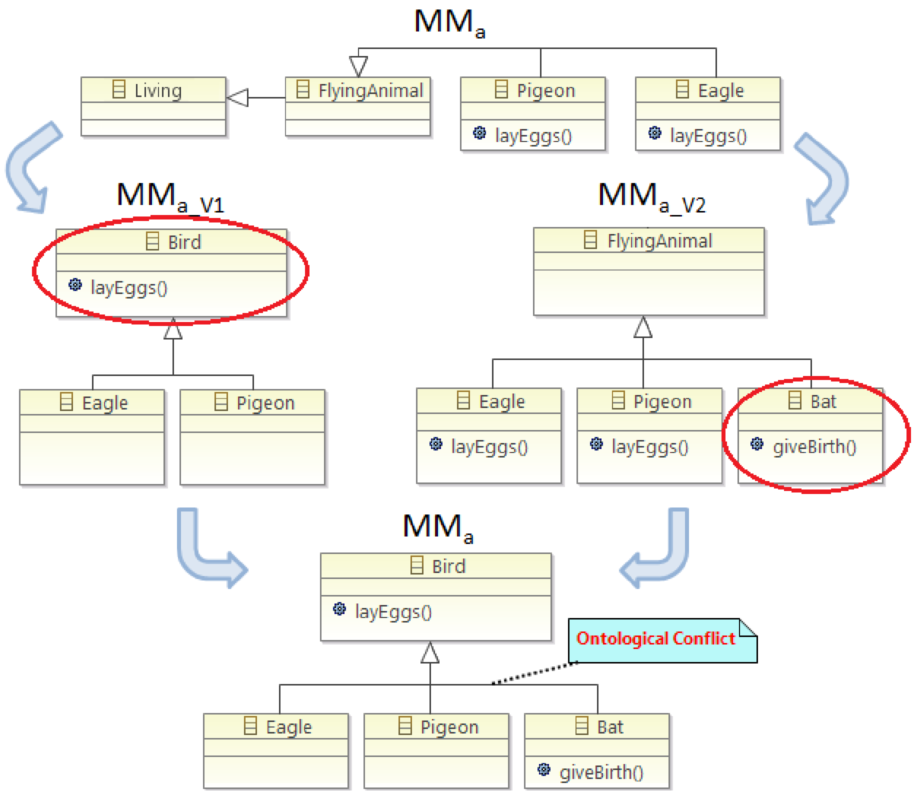

Fig. 1. Motivating Scenario

a valid ontological instance of the type `person`. It is worth noting that while linguistic aspects are invariants of the modelling activity, the ontological part is strictly coupled with the domain taken into account, and hence the purpose the modelling activity is devoted to. More importantly, ontologies implicitly define a set of semantic relationships which would need to be explicitly specified otherwise, as proposed in existing works on semantic model versioning [3]. Therefore, the semantics can be considered as direct consequence of adopting a given ontology for the domain taken into account. The idea of separating linguistic and ontological matters to reduce the complexity of modelling management, and/or to enhance reuse chances, is not new. In general these techniques have been referred to as deep or multilevel metamodelling [4, 5], to stress the fact that it could be useful to consider more than two (fixed) metamodelling layers. Based on the partition of linguistic and ontological aspects, it has been possible to support generic modelling language and transformation specifications [6, 7], and to lower the complexity of language evolution [8]. Notably, even if by exploiting different terminologies, all the mentioned works leverage on the distinction between linguistic and ontological aspects to define generic operations that are later on applied to the metamodel taken into account. In particular, linguistic manipulations can be replicated directly, while ontological ones have to be bound to the concepts pertaining to the considered applicative domain. This allows to create operators over models, define constraints, model transformations, and manage the need for language evolution. In [9] this methodological approach is the foundation for a framework supporting generic model management operators, which have been implemented within the Epsilon family of (meta-)modelling languages [10]. A similar solution is developed by means of the Melanie tool [11]; also in this case the modelling environment uses a multilevel modelling approach. Moreover, the ATLAS Transformation Language has been extended to encompass predicates distinguishing between the various modelling levels.

Despite the growing research interest and effort in this area, so far there has been little effort in the definition of evolution management support based on the separation between linguistic and ontological aspects. In this respect, with this paper we propose to revisit the main model versioning features in terms of such a separation to improve their efficacy. In order to better grasp the potentials of this idea, in Fig. 1 a sample evolutionary scenario is depicted. Metamodel $\mathtt{MM}_a$ defines a language for the definition of living creatures with focus on flying animals. As it can be noticed in its original form the metaclass `FlyingAnimal` is specialised by the sub-types `Eagle` and `Pigeon`, both containing a `layEggs` operation. Let us now suppose that $\mathtt{MM}_a$ is exposed, and concurrently evolves, in two different views, resulting in two versions of $\mathtt{MM}_a$, namely $\mathtt{MM}_{a\_V1}$ and $\mathtt{MM}_{a\_V2}$. Both view-specific metamodels undergo modifications. In case of $\mathtt{MM}_{a\_V1}$, the metaclass `FlyingAnimal` is renamed to `Bird` and the operation `layEggs` is moved from the sub-types to the super-type. In $\mathtt{MM}_{a\_V2}$, the new sub-type `Bat` is added together with its operation `giveBirth` as specialisation of `FlyingAnimal`. These modifications result in an ontological conflict from the perspective of $\mathtt{MM}_a$, since a bat is both a living creature and a flying animal giving birth to live young but NOT a bird laying eggs.

3 A Research Agenda

In the latest years a considerable research work has been devoted to all the activities involved in evolution management, notably *model differencing*, *conflict management*, as well as *metamodel evolution* and *model co-evolution*. Providing a survey on all those investigations goes far beyond the scope of this work, however in the next sections we outline some common principles and problems characterizing the current available solutions. In general current approaches for model versioning that can be found in the literature have to fight intrinsic semantics issues entailed by the modelling level of abstraction.

This paper aims at providing the guidelines for a novel version management methodology that takes into consideration the separation between linguistic and ontological aspects involved in modelling activities. Our belief is that, by means of such a separation, domain-specific issues can be better managed thus improving degree of automation and accuracy of current version management. In this respect, the next sections also illustrate a research agenda to revisit current versioning solutions based on the separation between linguistic and ontological aspects, discussing foreseeable benefits and needs.

At this point, it is worth noting that this proposal is not excluding the current available techniques, whereas it provides additional means to better exploit those solutions. By embracing the MDE principles, versioning artefacts are models conforming to corresponding metamodels and are manipulated by means of model transformations [2]. In this respect, this work relies on a model-based representation of differences as the ones proposed in [12, 13]. In particular, we exploit the difference representation proposal in [12] because of its generative approach that allows to adapt the already existing solutions to our separation in a smooth way.

The general approach we pursue is composed by two main constituents, a generic mechanism tackling the structural part and its evolution, and a domain-specific specialization that is bound to the ontology taken into account. As a consequence, general evolution patterns are described in linguistic terms, while ontological information is exploited to refine their management.

3.1 Model Differencing

Model differencing has been firstly addressed by means of language-specific solutions and then generalized to language-agnostic cases. Typically, difference detection, representation, and visualisation are performed at model level of abstraction for being able to grasp user's intentions [14, 15]. *State-based* techniques deduce modification operations based on the old and new state of a model [16]. The element matching can result very complex and hard to make arbitrarily precise, since it is reduced to the graph homomorphism problem [17]. A way to reduce such an inherent intricacy is to adopt *operation-based* approaches which keep track of the operations performed by the users to modify the model [18]. The drawback of such approaches is that the differences detection is tightly coupled to the tool, since anything happening outside the tool cannot be tracked in terms of evolution information.

Regardless being state- or operation-based, differencing approaches rely on structural similarities to determine evolution operations, thus requiring user intervention whenever the semantics involved in the changes is misinterpreted or cannot be grasped at all. Notably, if we consider the example shown in Fig. 1, a differencing engine would detect a rename of the `FlyingAnimal` class towards `Bird` since their subgraphs are matching from a structural perspective. However, this information does not provide any additional detail about the domain-specific consequences of such a modification.

In our vision, we propose to exploit ontological information as part of the differencing detection and result. By adopting a model-based mechanism like the one proposed in [12], each metaclass of the structural part involved in the modelling activity can be extended with corresponding evolution means as shown in part Figure 2.A. In this way, we are able to express the generic concepts for model evolution without coupling this information with domain-specific details. In relation to the scenario depicted in Fig. 1, the structural differencing would detect the addition of a class named `Bat` together with its operation `giveBirth` (see Figure 2.B). Analogously, the renaming of `FlyingAnimal` to `Bird` would be detected as shown in the upper part of Figure 2.C. On the contrary, the moving of the `layEggs` operation would be represented as a simple deletion in `Pigeon` and `Eagle` classes and as an addition in the `Bird` class (central and bottom parts of Fig-

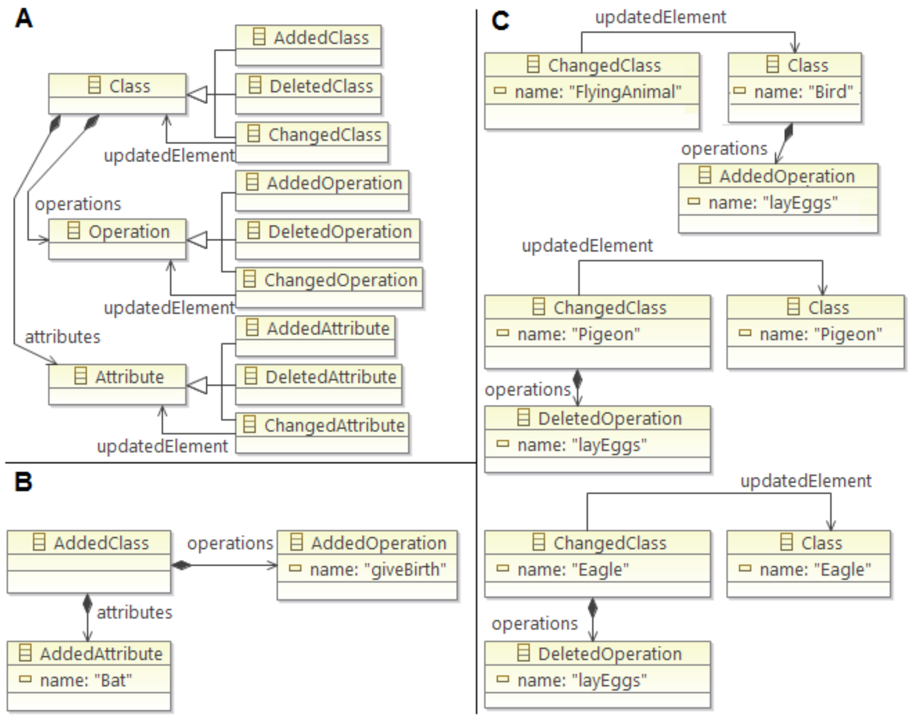

Fig. 2. Difference Metamodel for an Ecore-like Formalism (A) and Structural Differences from $\mathtt{MM}_a$ to $\mathtt{MM}_{a\_V1}$ (B) and $\mathtt{MM}_a$ to $\mathtt{MM}_{a\_V2}$ (C)

ure 2.C). It is worth noting that, in this difference detection step no semantics matters are involved and the evolution is observed from a pure linguistic point of view.

Domain-specific (or ontological) information instead is bound later on to provide additional support to the detection mechanism. Notably, by taking into account the information coming from a selected ontology (like the one depicted in Figure 3) in the situation mentioned above, it would be possible to obtain a customized difference algorithm in which `layEggs` is correctly detected as moved from the two `Pigeon` and `Eagle` subclasses to `Bird`. Consequently, differences would not only consider modifications from a structural point of view, but also the information related to the ontological evolution. Those details can be useful for detection and visualization purposes: the ontological relationships among elements could help in distinguishing between a rename and a delete/add evolution and to show changes from a domain-specific perspective, respectively. Moreover, they become very relevant when dealing with version merging and/or co-evolution management, as discussed in the remainder of the paper.

3.2 Conflict Management

Conflict management followed the same development differencing techniques did, that is detection and resolution have been addressed firstly at atomic operation level [19],

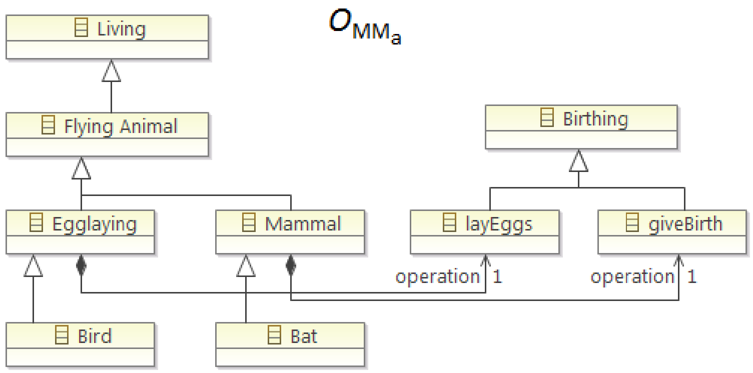

Fig. 3. Ontological Metamodel

then by considering refactoring modifications [20], and eventually by supporting arbitrary semantics divergences [21, 22]. Also in this case, and possibly even more important than for differencing, when domain aspects get involved in the management of concurrent modifications the precision of structure-based solutions degrades remarkably and user intervention is unavoidable.

By referring to the example illustrated in Fig. 1, a linguistic merging operation would not reveal any problem, since the involved subgraphs structures are perfectly compatible. However, by taking into account also ontological information it is possible to discover deeper issues. In particular, `Bat` becomes a specialization of `Bird` and `layEggs()` is inherited in the `Bat` class, thus arising an ontological conflict. It is very important to notice that, while the latter conflict could be solved by structural constraints (e.g., only one operation per class is admitted), the former has to be explicitly defined by the user. Even more important, in both cases the ontological aspects disclose the possibility to grasp the rationale behind the problems: `Bat` is not a `Bird` since, being a mammal, it does not `layEggs()` (see the ontology definition in Figure 3). Additionally, ontology information provides an hint on how to solve the problem: in fact, by preserving the `FlyingAnimal` class as parent of both `Bird` and `Bat` the conflict would be reconciled (as depicted in Fig. 4). From a conflict management perspective, the separation between linguistic and ontological aspects discloses very interesting research directions. Generic conflict detection and resolution strategies can be provided as based on linguistic aspects, and later on specialized taking into account ontological information. In this respect, while linguistic conflicts have to be solved since they affect the well-formedness of the merge result itself, ontological divergences can be *tolerated*. Therefore the separation proposed in this paper could be very useful, especially in the early stages of development, to allow collaborative development without forcing the users in taking domain-specific design decisions when their side effects are not completely clear [23].

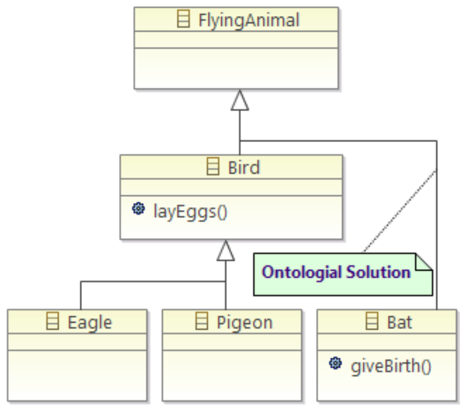

Fig. 4. Conflict Reconciliation

3.3 Metamodel Evolution and Model Co-Evolution

In MDE, metamodels are subject to the same evolutionary pressure models do. Metamodel evolutions trigger model co-evolutions, i.e. model instances have to be migrated to the newer version of the metamodel in order to recover their conformance [24]. In this scenario a correct interpretation of metamodel manipulations is of critical importance to adopt appropriate migration countermeasures. Moreover, model co-evolution may require user information to resolve particular migration cases [25]. As a consequence, a number of approaches have been introduced, supporting from (semi-)automated to manual model co-evolution approaches [26].

In the case of metamodel evolution domain-specific issues can heavily affect the migration process, especially when the whole metamodel ecosystem is involved in the evolution [27]. Therefore, recent investigations have been devoted to relax the metamodel-model conformance relationship, even by separating linguistic and ontological aspects involved in the language definition [8]. Our idea is based on the same principle of this latter work, but instead of relaxing the conformance relationship we propose to separate linguistic and ontological aspects and address their co-evolution separately. Analogously to the model merging problem, also in this case structural co-evolution has to be performed in order to re-establish the linguistic well-formedness. Whereas, ontological issues can be solved in a separate way and by means of domain-specific solutions.

It is not expectable that the separation between linguistic and ontological aspects will guarantee full automation of co-evolution operations. However, by knowing the metamodel evolution in ontological terms can help in managing it in a better way. In particular, detecting `layEggs()` as a moving operation rather than a delete/add manipulation would avoid loss of information in the migration stage. Moreover, by noticing that `Bat` can not specialise `Bird` for the before-mentioned reasons defined in the ontology, a migration operation would add a new metaclass `Bird` instead or renaming `FlyingAnimal` (see Figure 4). In turn, `Bat` would be kept as it is after the migration being still a valid instance of `FlyingAnimal`. Interestingly, since `Bat` is still a `FlyingAnimal` a tool co-evolution countermeasure could also decide to re-use the same icon, or ask for a new one specific for bats. In the same way, it could be possible to notice that using the `Bird` icon would be erroneous from a domain semantics perspective.

4 Outlook

This paper proposed the guidelines for a novel model versioning methodology based on the separation between linguistic and ontological aspects of (meta-)modelling. The idea is not ignoring what already existing and developed in the latest years for model evolution investigations; rather, it aims at enriching current solutions by adding ontological details to the manipulation information. In this respect, we remark once again that current semantics aware versioning solutions do not aim at the clear separation between structural and ontological aspects, which typically get intertwined in the metamodel definition. Moreover, we consider the ontological part as domain information to be *plugged-in* in the generic version management mechanism. Such approach allows to build-up generic differencing, merging, and co-evolution techniques, taking into account ontological information as a refinement step.

Up to now, it has been possible to conduct small experiments by means of already available techniques (notably [12] and the Melanie tool [11]) and the results are encouraging. However, the methodology has to be validated against real-life systems to prove its efficacy. Moreover, it is not possible to exclude future needs for addressing ontological-specific evolutions, both for the detection, representation, and management, beyond the general additions, deletions, and changes.

References

1. Kühne, T.: Matters of (meta-)modeling. SoSym **5** (2006) 369–385
2. Bezivin, J.: On the Unification Power of Models. SoSym **4** (2005) 171–188
3. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In: Procs. of MoDELS, Genova (Italy). LNCS, Springer (2006) 528–542
4. de Lara, J., Guerra, E.: Deep Meta-modelling with MetaDepth. In: Proc. of TOOLS, Málaga (Spain). LNCS (2010) 1–20
5. Atkinson, C., Gutheil, M., Kennel, B.: A Flexible Infrastructure for Multilevel Language Engineering. IEEE TSE **35** (2009) 742–755
6. de Lara, J., Guerra, E., Cuadrado, J.S.: Abstracting Modelling Languages: A Reutilization Approach. In: Proc. of CAiSE, Gdansk (Poland). LNCS, Springer (2012) 127–143
7. Cuadrado, J.S., Guerra, E., de Lara, J.: Generic Model Transformations: *Write Once, Reuse Everywhere*. In: Procs. ICMT, Zurich (Switzerland), 2011. LNCS, Springer (2011) 62–77
8. Gómez, P., Sánchez, M., Florez, H., Villalobos, J.: Co-creation of models and metamodels for enterprise architecture projects. In: Procs. of XM, ACM (2012) 21–26
9. Rose, L.M., Guerra, E., de Lara, J., Etien, A., Kolovos, D.S., Paige, R.F.: Genericity for model management operations. SoSym **12** (2013) 201–219
10. : Epsilon. http://www.eclipse.org/epsilon/ (2013)
11. : Melanie - multi-level modeling and ontology engineering environment. http://code.google.com/a/eclipselabs.org/p/melanie/ (2013)

12. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: A Metamodel Independent Approach to Difference Representation. JOT **6** (2007) 165–185
13. Rivera, J., Vallecillo, A.: Representing and Operating with Model Differences. In: Procs. TOOLS EUROPE. (2008)
14. Kolovos, D., Paige, R., Polack, F.: Model comparison: a foundation for model composition and model transformation testing. In: Procs. GaMMa, Shanghai (China). (2006) 13–20
15. Brun, C., Pierantonio, A.: Model Differences in the Eclipse Modeling Framework. UPGRADE, The European Journal for the Informatics Professional (2008)
16. Conradi, R., Westfechtel, B.: Version Models for Software Configuration Management. ACM Computing Surveys **30** (1998) 232–282
17. Kolovos, D.S., Di Ruscio, D., Paige, R.F., Pierantonio, A.: Different models for model matching: An analysis of approaches to support model differencing. In: Proc. 2nd CVSM'09, ICSE09 Workshop, Vancouver, Canada (2009)
18. Mens, T.: A State-of-the-Art Survey on Software Merging. IEEE Trans. Softw. Eng. **28** (2002) 449–462
19. Alanen, M., Porres, I.: Difference and Union of Models. In: UML 2003 - The Unified Modeling Language. Volume 2863 of LNCS., Springer-Verlag (2003) 2–17
20. Mens, T., Taentzer, G., Runge, O.: Detecting Structural Refactoring Conflicts Using Critical Pair Analysis. Electr. Notes Theor. Comput. Sci **127** (2005) 113–128
21. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. International Journal of Web Information Systems (IJWIS) **5** (2009) 271 – 304
22. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: Managing Model Conflicts in Distributed Development. In: Procs. MoDELS. (2008) 311–325
23. Wieland, K., Langer, P., Seidl, M., Wimmer, M., Kappel, G.: Turning conflicts into collaboration. Computer Supported Cooperative Work **22** (2013) 181–240
24. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software **20** (2003) 42–45
25. Gruschko, B., Kolovos, D., Paige., R.: Towards Synchronizing Models with Evolving Metamodels. In: Procs of the Work. MODSE. (2007)
26. Rose, L.M., Herrmannsdoerfer, M., Williams, J.R., Kolovos, D.S., Garcés, K., Paige, R.F., Polack, F.A.C.: A Comparison of Model Migration Tools. In: Procs. MoDELS. LNCS, Springer (2010) 61–75
27. Iovino, L., Pierantonio, A., Malavolta, I.: On the impact significance of metamodel evolution in mde. Journal of Object Technology **11** (2012) 3: 1–33