Different Models for Model Matching: An analysis of approaches to support model differencing

Dimitrios S. Kolovos
Department of Computer Science
The University of York
dkolovos@cs.york.ac.uk

Davide Di Ruscio
Dipartimento di Informatica
Universita degli Studi dellAquila
diruscio@di.univaq.it

Alfonso Pierantonio
Dipartimento di Informatica
Universita degli Studi dellAquila
alfonso@di.univaq.it

Richard F. Paige
Department of Computer Science
The University of York
paige@cs.york.ac.uk

Abstract

Calculating differences between models is an important and challenging task in Model Driven Engineering. Model differencing involves a number of steps starting with identifying matching model elements, calculating and representing their differences, and finally visualizing them in an appropriate way. In this paper, we provide an overview of the fundamental steps involved in the model differencing process and summarize the advantages and shortcomings of existing approaches for identifying matching model elements. To assist potential users in selecting one of the existing methods for the problem at stake, we investigate the trade-offs these methods impose in terms of accuracy and effort required to implement each one of them.

1 Introduction

With the increasing adoption of MDE, the development of versioning mechanisms for supporting the evolution of model-based artefacts is becoming essential [23, 5]. System development and evolution processes require modifying the involved artefacts several times. Consequently, nurturing the detection of differences between models is essential to model development and management practices.

In an MDE setting, the document comparison algorithms provided by version control systems like CVS [8], and SVN [7] have been shown to be inadequate since they compare models at a particularly low level, and lack a reasonable organization and abstraction from designer's perspective [13]. In this respect, there has been intense research in the field of model comparison, especially for UML diagrams [1, 15, 24]. Recently, a number of works have proposed generalizations of such approaches in order to compare models conforming to any arbitrary metamodel [12, 18].

Calculating model differences is a difficult task since it relies on *model matching* which can be reduced to the graph isomorphism [16]: the problem of finding correspondences between two given graphs. Theoretically, the graph isomorphism problem is NP-hard [12] and the available approaches tend to deal with the computational complexity by providing solutions which are modeling language specific or are able to approximate the exact solution.

In this paper, we analyse a number of existing model matching approaches and evaluate them against several requirements such as accuracy, efficiency, tool and domain independence. Based on the results of our analysis, we argue that there is no single *best* solution to model matching but instead that the problem should be treated by deciding on the best trade-off within the constraints imposed in the context, and for the particular task at stake.

The paper is structured as follows. Section 2 introduces the problem of model differencing and examines it under different perspectives. Section 3 describes the most significant model matching approaches which are then compared in the context of a common case study in Section 4.Finally, Section 5 concludes the paper and provides directions to further work on the subject.

2 Model differencing in MDE

The problem of determining model differences is intrinsically complex. The overall problem can be separated into three phases [2]:

– *calculation*, a procedure, method or algorithm able to compare two distinct models;

– *representation*, the outcome of the calculation must be represented in some form which is amenable to further manipulations;
– *visualization*, model differences are often required to be visualized in a human-readable notation which enables the designer to grasp the rationale behind the modifications which the models have undergone.

The following paragraphs provide a brief discussion on existing approaches to each one of these phases.

Calculation The task of model comparison consists of identifying the mappings and the differences between two models. In the context of software evolution, difference calculation has been intensively investigated as witnessed by a number of approaches ranging from text comparisons to model differencing techniques. Specialized differencing methods, such as [1, 15, 24], have been introduced to strictly compare UML models. A generalization of the work by Xing and Stroulia [24] is proposed in [12] which presents an approach based on structural similarity which is able to compare not only UML diagrams but also models conforming to arbitrary metamodels. The problem of calculating differences between models conforming to arbitrary metamodels, has been taken into account also by [18] which propose a comprehensive modeling management environment based on the Maude language [6].

Representation The information obtained from the difference calculation step needs to be properly represented in a difference model, so that it can be used for subsequent analysis and manipulation [4]. Finding a suitable representation for model differences is crucial for its exploitation, as for instance deriving refactoring operations from a delta document describing how a database schema evolved in time. The effectiveness of model difference representations is often compromised by factors such as the calculation method or the scope of the model difference. For instance, in the case of edit scripts [1, 13] the representation is operational since it describes how to modify the initial model in order to obtain the final model. Such a representation notation requires ad-hoc tools and suffers from a lack of abstraction by reducing the manipulation or analysis possibilities. In other cases, the representation may be model based and enable automatic manipulation of the differences, as in the case of coloring [15], but the visualization and the representation tend to overlap and the overall method is affected by the way the differences are computed.

Visualization Differences often need to be presented according to a specific need or scope, highlighting those pieces of information which is relevant only for the prescribed goal. In other words, a visualization is realized by specifying a concrete syntax which renders the abstract syntax (representation) and may vary from intuitive diagrammatic notations to textual catalogues (e.g. spreadsheet data). The same representation may include different visualizations, not necessarily diagrammatic ones, depending on the specific purpose the designer has in mind. In this respect, both edit scripts and coloring represent two different visualizations although they are generated directly by the specific differencing algorithm and letting the representation be rendered by means of internal formats which prevent them from being processed in tool chains. For instance, edit scripts render both representation and visualization with the same notation.

Calculation and representation are the central ingredients for any model comparison solution. In the rest of the paper we focus on the former aspect, while the latter is important in tool chaining and user readability. A number of difference calculation approaches is analysed with respect to the most intricate part of the calculation task: model matching.

3 Current approaches to model matching

As discussed in [19], there are several requirements for model matching approaches including accuracy, a high level of abstraction at which comparison is performed, independence from particular tools, domains and languages, efficiency, and minimal effort. In the following sections we provide a review of existing approaches to model matching and demonstrate that those requirements can each other. For example, a language-specific matching algorithm is likely to achieve better accuracy and performance than a generic similarity-based algorithm, but on the other hand requires a significant amount of effort to implement. As a result, we argue that there is no single *best* solution to model matching but instead that the problem should be treated as deciding on the best trade-off within the constraints imposed in the context, and for the particular task at stake. Table 1 provides a compact overview of the main existing approaches and summarizes their features.

3.1 Static Identity-Based Matching

In this approach, it is assumed that each model element has a persistent and non-volatile unique identifier that is assigned to it upon creation. Therefore, a basic approach for matching models is to identify matching model elements based on their corresponding identities (as in [1, 11]). The main advantages of this approach are that it requires no configuration from the user perspective and that it is particularly fast. On the other hand, this approach does not apply to models constructed independently of each other, and to model representation technologies that do not support maintenance of unique identities.

	Static identity-based	Signature-based	Similarity-based	Customization support
Alanen and Porres [1]	Language specific	-	-	-
DSMDiff [12]	-	-	Language independent	-
EMFCompare [10]	-	-	Language independent	Custom matching algorithms
ECL [9]	-	-	Language independent	DSL for specifying custom matching rules
Melnik et al. [20]	-	-	Language independent	Custom filters and parameters
Nejati et al. [14]	-	-	Language specific	-
Reddy et al. [17]	-	Language independent	-	-
Rivera et. al [18]	-	-	Language independent	-
SiDiff [22]	-	-	Language independent	Weight configuration, custom algorithms for similarity calculation
TOPCASED [11]	Language independent	-	-	-
UMLDiff [24]	-	-	Language specific	-

Table 1. Model matching approaches

3.2 Signature-Based Matching

In [17], the authors recognize the limitations posed by static identity-based matching and propose signature-based matching instead. In this technique, the identity of each model element is not static, but instead it is signature calculated dynamically from the values of its features by means of a user-defined function specified using a model querying language (e.g., see [17]). As this approach does not rely on persistent identities it can be also used to compare models that have been constructed independently of each other. Unfortunately, this additional benefit comes at a price: while in the static-identity matching approach no configuration effort is required, in this approach developers need to specify a series of functions that calculate the identities of different types of model elements.

3.3 Similarity-Based Matching

While the previous two approaches treat the problem of model matching as true/false identity (whether static or dynamic) matching, this category of approaches treats models as typed attribute graphs and attempts to identify matching elements based on the aggregated similarity of their features. However, not all features of model elements are equally important for model matching (e.g. classes with matching names are more likely to be matching than classes with matching values in their *abstract* feature). Therefore, similarity-based algorithms typically need to be provided with a configuration that specifies the relative *weight* of each feature. Typical examples of this category of approaches are SiDiff [22], the approach presented in [21], the similarity flooding algorithm presented in [20], and DSMDiff [12] (which also incorporates signature based matching). SiDiff provides also the means to specify custom algorithms for computing the similarity of properties. The built-in algorithm of EMF Compare [10] also falls within this category but provides a fixed configuration. Compared to identity-based matching, typed attribute graph matching algorithms have been shown to produce more accurate results. On the other hand, establishing and fine-tuning the weights of the features is a predominately empirical trial-and-error process, and as such, finding the exact values of weights that deliver the best results for a particular modelling language can be particularly challenging. Moreover, by being generic, such approaches fail to take into consideration the semantics of the modelling language which, as shown below in Section 3.4, if taken into consideration can improve both the accuracy of the results, and also significantly reduce the number of individual comparisons (search space).

3.4 Custom Language-Specific Matching Algorithms

This category involves matching algorithms tailored to a particular modelling language such as UMLDiff [24] and the work in [14] which specifically target UML models and statecharts, respectively. The main advantage of a language-specific matching algorithm is that it can incorporate the semantics of the target language in order to provide more accurate results, and also drastically reduce the search space too. For instance, when comparing UML models, a UML-specific matching algorithm can exploit the fact that two classes or data types with the same name constitute a match – for all practical reasons – regardless of their location in the package structure, while the same does not hold for other types of elements (such as parameters or operations). Moreover, it can incorporate the knowledge that it only makes sense to compare two operations if the classes they belong to are already known to match - and similarly for properties and parameters -, thus drastically reducing the number of comparisons that need to be performed. Also, if a single in-

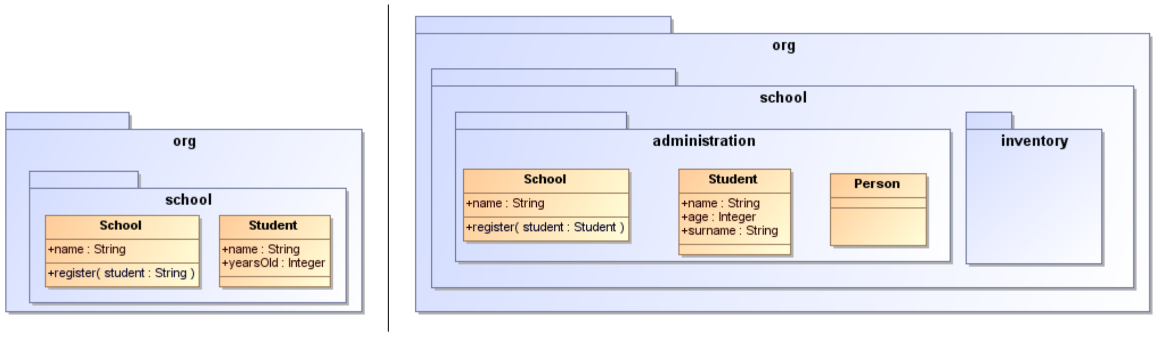

Figure 1. Different versions of a sample UML model

heritance idiom is adopted – which is the standard practice when UML is used to model a system that will later on be implemented in a single-inheritance language such as Java – the algorithm can compare generalizations based only on the value of their *specific* feature, while disregarding the value of their *general* feature.

However, all these advantages come at a high price. While for previously discussed approaches developers need to spend little (e.g. provide a configuration or write signature generators) or no effort at all (e.g. identity matching), under this approach, they need to specify the complete matching algorithm manually, which can be a particularly challenging task. To ease the development of custom matching algorithms, approaches such as EMF Compare and the Epsilon Comparison Language (ECL) [9] provide infrastructure that can automate the trivial parts of the comparison process, allowing developers to concentrate on the comparison logic only. Nevertheless, even with such tool support, the effort required to implement a custom matching algorithm is still considerable.

4 Case Study

In this section we propose a brief demonstration of how different model matching approaches behave on a common example. In particular, the sample UML models reported in Figure 1 are considered throughout the section. The models have been kept deliberately simple because of space limitations, but we believe they are sufficient to indicate how the concepts outlined in the previous section apply to concrete models.

The refactoring operations which have been performed on the model in Figure 1.a leading to the one in Figure 1.b can be summarized as follows:

1. the classes `School` and `Student` have been moved to a new package `administration`;
2. the parameter `student` of the operation `register` in the class `School` has been modified by changing its type from `String` to `Student`;
3. the attribute `surname` has been added in the class `Student`
4. the attribute `yearsOld` has been renamed to `age`;
5. the class `Person` has been added in the new package `administration`;
6. the package `inventory` has been added in the package `school`.

Static identity-based matching approaches like [1, 11] are able to detect all the modifications previously itemized without any user effort. Even the renaming of the attribute `yearsOld` can be correctly discovered as depicted in Figure 2 which reports a fragment of the differences calculated by the TOPCASED tool [11].

As discussed in Section 3, static identity-based matching approaches reduce their applicability to the models which maintain the persistent and non-volatile unique identifiers assigned to the contained elements upon creation. In several cases this can be a too strong restriction. For instance, to compare the models in Figure 1 if they are created by means of different tools the adoption of signature-based matching approaches is preferable. EMFCompare [10] is one of the possible approaches that can be used in this case. It relies on a metamodel independent matching algorithm and by using it without any extension to compare the models in Figure 1, the differences in Figure 3 are returned. Since the standard matching algorithm is not aware of the UML semantics is not able to detect all the changes we expect. For instance, the modifications to the `School` and `Student` classes have been detected as deletions and additions of new ones with new structural features.

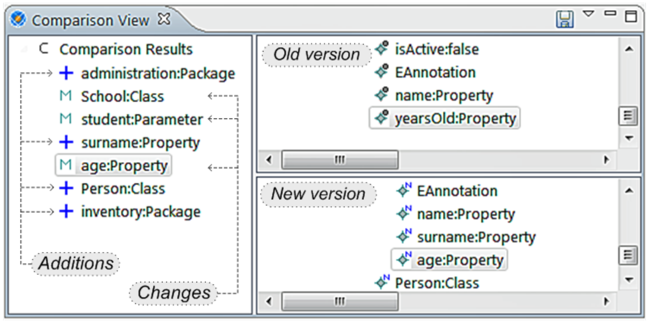

Figure 2. Fragment of the differences calculated by TOPCASED tool

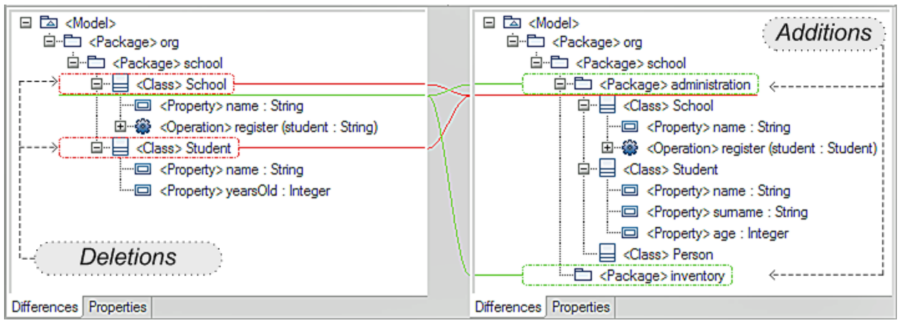

Figure 3. Differences calculated by EMFCompare

Depending on the problem at stake, the available time, and the accuracy one wants to achieve, the user might want a better comparison and in this case more information has to be provided to the matching algorithm. In this respect, SiDiff [22] can be used. As said in the previous section, it is a metamodel independent approach, hence in order to deal with the model in Figure 1, a custom configuration has to be provided by the users in order to manage UML models. Figure 4 reports the differences calculated by SiDiff configured with the settings available in the Fujaba tool suite [3] implementation. All the differences have been correctly detected but the renaming of the attribute `yearsOld`. In fact, SiDiff detects this modification as a deletion of the `yearsOld` attribute and an addition of the `age` one. This can not be considered an error of the approach since there is not a match between them that can be specified in general with respect to the semantics of UML.

Finally, if users need to specify a precise language-specific comparison algorithm that incorporates the semantics of the targeted language, they can use solutions such as ECL, or build a custom matching engine atop the infrastructure provided by the EMF Compare framework. For the specific example, we have implemented a rule-based comparison algorithm using ECL where, among others, we explicitly specified a rule that matches two attributes belonging to matching classes even if they don't have the same name, as long as their types match and they are also the only attributes of this type within their respective classes. The ECL-based approach also allowed us to reduce significantly the search space where this was reasonable to do. For example, while we compare all classes of the one model with all classes of the other model irrespectively of their location in the model structure, we only compare attributes and operations that belong to matching classes, and similarly, parameters that belong to matching operations. As a result, the `yearsOld` and `age` attributes were successfully matched as shown in Figure 5. Nevertheless, as discussed earlier, this was the most effort-consuming solution too, requiring 120 lines of ECL code only for a small subset of the UML2 metamodel.

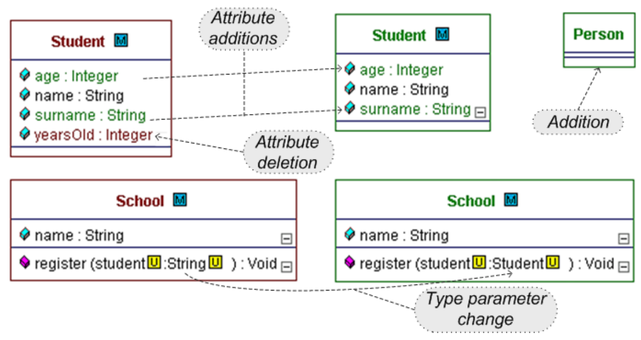

Figure 4. Differences calculated by SiDiff

5 Conclusions

From the discussion provided above, it becomes evident that selecting a model matching approach for the problem at stake involves deciding on a trade-off between the required accuracy and the effort necessary to accomplish the differencing. Therefore, when devoting effort to configuring/implementing a matching algorithm is not an option, a combination of an identity and similarity-based algorithm such as DSMDiff, SiDiff, or the built-in algorithm of EMF Compare is a fair trade-off. However, if some effort can be allocated to the task, establishing and fine-tuning a sensible configuration for a similarity algorithm, or implementing a custom signature-based approach is likely to pay off in the form of more accurate results. Finally, in the case that improved accuracy and performance justify allocating significant effort to the task, a custom matching algorithm based on infrastructure such as EMF Compare or ECL is deemed more appropriate.

Our arguments are based on practical experience obtained through experimentation with several implementations of matching algorithms and tools. However, we do not currently have directly comparable data for the different approaches, mainly because of the diversity in terms of the modelling technologies different algorithms and tools are implemented atop. For instance, the available implementation of SiDiff is built on top of Fujaba and appears to be supporting UML 1.3 models only. DSMDiff on the other hand has been built on top of GME, while ECL and EMF Compare operate on EMF models.

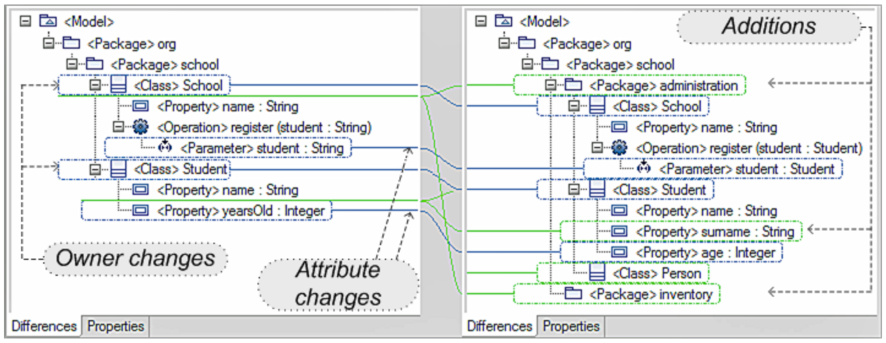

Figure 5. Differences calculated by ECL

In order to obtain directly comparable data regarding the accuracy of each algorithm and the effort required to customize it for the problem at stake, convergence in terms of the modelling technology on which different tools operate is essential. With the growing acceptance of EMF as the defacto standard in the Model Driven Engineering community we expect that actively maintained and used tools and algorithms will inevitably move towards this direction, and thus a comparison based on concrete results should be feasible in the not too distant future.

Acknowledgements This work is partially supported by the MODELPLEX (http://www.modelplex.org) project, co-funded by the EC under the "Information Society Technologies" Sixth Framework Programme (2006-2009) and by the EU MANCOOSI (http://www.mancoosi.org) project. We would like to thank Antonio Cicchetti for his insightful discussions on this work.

References

[1] M. Alanen and I. Porres. Difference and Union of Models. In *UML 2003 - The Unified Modeling Language*, volume 2863 of *LNCS*, pages 2–17. Springer-Verlag, 2003.

[2] C. Brun and A. Pierantonio. Model Differences in the Eclipse Modeling Framework. *UPGRADE, The European J. for the Informatics Professional*, April-May 2008.

[3] S. Burmester, H. Giese, J. Niere, M. Tichy, J. P. Wadsack, and L. Wendehals. Tool integration at the meta-model level within the fujaba tool suite. pages 51–56, 2003.

[4] A. Cicchetti, D. Di Ruscio, and A. Pierantonio. A Metamodel Independent Approach to Difference Representation. *JOT*, 6(9):165–185, October 2007.

[5] A. Cicchetti, D. Di Ruscio, and A. Pierantonio. Managing model conflicts in distributed development. In *MoDELS '08*, pages 311–325. Springer-Verlag, 2008.

[6] M. Clavel, S. Eker, and P. Lincoln. Maude: specification and programming in rewriting logic. 285:2002, 1999.

[7] B. Collins-Sussman, B. Fitzpatrick, and C. Pilato. *Version Control with Subversion. For Subversion 1.1.* O'Reilly & Associates, Inc., 2004.

[8] CVS Project. CVS web site. http://www.nongnu.org/cvs.

[9] D. S. Kolovos and R. F. Paige and F. A.C. Polack. Model Comparison: A Foundation for Model Composition and Model Transformation Testing. In *GaMMa'06*, pages 13–20. ACM Press, 2006.

[10] Eclipse Foundation. EMF Compare, 2008. http://www.eclipse.org/modeling/emft/ ?project=compare.

[11] P. Farail, P. Gaufillet, A. Canals, C. L. Camus, D. Sciamma, P. Michel, X. Crgut, and M. Pantel. The TOPCASED project: a Toolkit in OPen source for Critical Aeronautic SystEms Design. In *ERTS06*, 2006.

[12] Y. Lin, J. Gray, and F. Jouault. DSMDiff: A Differentiation Tool for Domain-Specific Models. 16(4):349–361, August 2007. (Special Issue on Model-Driven Development).

[13] T. Mens. A State-of-the-Art Survey on Software Merging. *IEEE Trans. Softw. Eng.*, 28(5):449–462, 2002.

[14] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *ICSE '07*, pages 54–64. IEEE Computer Society, 2007.

[15] D. Ohst, M. Welle, and U. Kelter. Differences between versions of UML diagrams. In *ESEC/FSE*, pages 227–236. ACM Press, 2003.

[16] R. C. Read and D. G. Corneil. The graph isomorphism disease. 1(4):339–363, 2006.

[17] R. Reddy, R. France, S. Ghosh, F. Fleurey, and B. Baudry. Model composition - a signature-based approach. In *AOM Workshop*, 2005.

[18] J. E. Rivera and A. Vallecillo. Representing and operating with model differences. LNBIP 11, Springer.:141–160, 2008.

[19] S. Fortsch and B. Westfechtel. Differencing and Merging of Software Diagrams, State of the Art and Challenges. In *ICSOFT*, 2007.

[20] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and ist application to schema matching. In *ICDE*, page 117128, 2002.

[21] P. Selonen and M. Kettunen. Metamodel-Based Inference of Inter-Model Correspondence. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, 2007.

[22] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference computation of large models. In *ESEC/FSE*, pages 295–304, 2007.

[23] A. van Deursen, E. Visser, and J. Warmer. Model-Driven Software Evolution: A Research Agenda. In *MoDSE'07*, pages 41–49.

[24] Z. Xing and E. Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In *ASE'05*, pages 54–65. ACM, 2005.