MODEL DIFFERENCES IN THE ECLIPSE MODELING FRAMEWORK

Cédric Brun

Obeo

44400 Reze, France

Alfonso Pierantonio

Dipartimento di Informatica

Università degli Studi dell'Aquila

I-67100 L'Aquila, Italy

Abstract. Increasingly, recording the various kinds of design-level structural evolution that a system undergoes throughout its entire life-cycle is gaining a fundamental importance and cannot be neglected in software modeling and development. In this respect, an interesting and useful operation between the designs of subsequent system versions is the difference management consisting in calculation, representation, and visualization. This work presents EMF Compare, an approach to model difference *calculation* and *representation* for the Eclipse modeling framework. Apart from leveraging the rank of model differences to that of first-class artifacts according to the "everything is a model" principle, the approach presents several properties which are discussed according to a conceptual framework.

1. Introduction

Last decade witnessed a dramatic growth of software intricacy and different techniques and methodologies have been proposed to ease complex system development. Model Driven Engineering (MDE) [S03] shifts the focus of software development from coding to modeling and let software architects harness the opportunity of dealing with higher-level of abstractions. In this respect, models represent descriptions of phenomena of the real (or imaginary) world which are usually complete w.r.t. designer’s goal, i.e. a specific task the designer is pursuing such as code generation or software analysis. However, models reach their fundamental effectiveness when they can be manipulated by means of automated transformations in order to obtain different kinds of artifacts ranging from other models to documentation or even implementation code. It is important that designers are able to comprehend the various kinds of design-level structural evolution that a software system undergoes throughout its entire life-cycle. Nurturing the detection of differences between models is essential to model development and management practices, which are traditionally not neglected in high-quality software development processes [CW98]. Thus, these activities are crucial not only for understanding the system design and its evolution but also for obtaining an accurate picture of the quality requirements of the system so that it can be consistently evolved.

The problem of model differences is intrinsically complex and requires algorithms and notations [KPP06, LZG04] which permit to fully profit from its potential in MDE. The paper presents part of the state of the art in calculating model differences and outlines a conceptual framework which prescribes crucial requirements to leverage differences to first-class entities. Accordingly, a solution must necessarily have a high degree of separation between three relevant aspects in model differencing: *calculation*, *representation*, and *visualization*. In fact, in current proposals the distinction between the three aspects are often blurred compromising the adoption of generic modeling techniques [B05]. In this paper, we discuss the problem of model differences and illustrate how EMF Compare [EC08] addresses this difficult task in the Eclipse generic platform. In particular, the approach is metamodel independent, i.e. it does apply to models which conform to arbitrary metamodels, and based on similarity techniques (see Sect. 2) which provide enhanced flexibility and interoperability. Moreover, it is model-based in the sense that the outcome of a model comparison is represented by means of a model which enables its manipulations in model-to-model or model-to-text transformations.

The paper is structured as follows: in Sect. 2 an introduction to the problem of model differences is presented and a number of representation requirements are given. Next Section presents EMF Compare describing both calculation, representation, and an evaluation w.r.t. the requirements introduced in Sect.2. Finally some conclusions are drawn.

2. **Model Difference**

As previously mentioned, the problem of model differencing is intrinsically complex and in order to analyse and/or propose a possible solution, it is important to decompose the problem in its constituting parts. In fact, its complexity is manifold and refers at least to the following aspects

a) *calculation*: a procedure, method or algorithm able to compare or differencing two distinct models;
b) *representation*: the outcome of the calculation must be represented in some form, current notations present deficiencies since they are heavily affected by the calculation method or by the application they are thought for;
c) *visualization*: model differences often requires to be visualized in a human-readable notation which let the designer grasp the rationale behind the modification the models underwent during their lifetime.

In the sequel of the paper we will discuss these aspects according to the available literature and will try to present and characterize EMF Compare according to them.

Calculation. 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. As stated by T. Mens in [M02], delta calculation algorithms can be classified by different points of view, each of which related to the particular application the approach is used for. Specialized differencing methods have been introduced to strictly compare UML diagrams, such as [AP03, OWK03, XS05] among others. These approaches can be divided in two main categories depending whether they make use of *persistent identifiers* or *similarity* metrics: the former heavily relies on identifiers which are assigned to model elements by the modeling tools which compromises interoperability and lock the models within a specific platform since identifiers are not universally computable; the latter establishes how similar two model elements are by comparing not only the properties local to the elements but also their neighborhood which makes the method agnostic of the modeling tools being independent from the any identification mechanism. A generalization of the work in [XS05] is an approach based on structural similarity which is able to compare not only UML models but also models conforming to any arbitrary metamodel [LGF07]. This represents an advance towards a wider acceptance about difference and version management in software development and generic modeling platforms (for instance [BJR04, LMB01]).

Representation. Detecting differences and identifying mappings among distinct versions of a system design is preparatory to represent at least part of such knowledge. 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. However, the effectiveness of representation of model differences is often compromised by a number of factors such as the calculation method or the scope the model difference has been thought for. For instance, in the case of edit scripts the representation is operational since it describes how to modify the initial model in order to obtain the final model. Clearly, such a representation notation suffers of a lack of abstraction and, let alone the capability of reconstructing the final model, does not easily allow any further manipulation or analysis since necessarily requires *ad-hoc* tools. In other cases, the representation may be even model-based (which permits further manipulations of the differences), as in the case of coloring, but the visualization and the representation tend to overlap and the overall method is affected by the way the differences are computed, i.e. in a set-theoretic fashion. In general, a proper representation must contain all the information defining the differences and make this knowledge available to further analyses and manipulations. Thus, we believe it must be given in terms of *abstract syntax* by introducing suitable metamodels as outlined afterwards.

Visualization. Differences often require to be presented according to a specific need or scope highlighting those pieces of information which are relevant only for the prefixed goal. In other words, a visualization is realized by giving a *concrete syntax* which pretty-prints the abstract syntax (representation) and may vary from intuitive diagrammatic notations to textual catalogs as, for instance, spreadsheet data. The same representation may give place to different visualizations 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 realized by means of internal formats which prevents them from being processed in tool chains. For instance, edit scripts realizes both representation and visualization with the same notation.

Clearly, the calculation and the representation are the central ingredients for any solution. In particular, we are interested in those representations which leverage model differences to the rank of first class objects fulfilling the “everything is a model” principle [B05]. As a consequence, a number of desirable properties must be imposed on a representation techniques as discussed in [CDP07] and described below

1) *model-based*, the outcome of a difference calculation must be represented as a model to enable a wide range of possibilities, such as subsequent analysis, conflict detection or manipulations;

2) *compactness,* the difference model must be compact and contain only the necessary information to represent the modifications, without duplicating parts as those model elements which are not involved in the change;
3) *self-contained,* a difference model must not rely on external sources of information, as for instance references to base model elements or base metamodels;
4) *transformative*, each difference model must induce a transformation, such that whenever applied to the initial model yields the final one. Moreover, the transformation must be applicable also to any other model which is possibly left unchanged in case the elements specified in the difference model are not contained in it;
5) *compositionality,* the result of subsequent or parallel modifications is a difference model whose definition depends only on difference models being composed and is compatible with the induced transformations;
6) *metamodel independence,* the representation techniques must be agnostic of the *base* metamodel, i.e., the metamodel the base models conform to. In other words, it must be not limited to specific metamodels, as for instance happens for certain calculation methods (e.g., [OWK03, XS05]) which are given for the UML metamodel.

The above discussion presents a minimal set of requirements which should be taken into account in order to let a generic modeling platform deal with an advanced form of model management. In the next section, we will illustrate EMF Compare showing how the approach fulfill most of the described requirements.

3. EMF Compare

EMF Compare is an Eclipse project initiated in 2006 at Eclipse Summit Europe, there the need for a model comparison engine emerged as an evidence, the Obeo and Intalio companies [Ob08,In08] contributed the first implementation of this component which has had two stable releases from that time. The goals of this component is to provide "out of the box" model comparison and merge support. Even if we think that one unique algorithm is able to provide good results both in term of efficiency and performances, we are aware that there may be several solutions to this problem, at different levels of generality and which depend on the main concerns one wants to address with model comparison (see Sect. 2). That's why this component has been designed with a high degree of extensibility in mind and every part of the whole comparison process is customizable.

The global comparison process is generally admitted as being composed of two main parts: the *matching* and the *differencing* part. In EMF Compare these parts are explicitly disjoint and done by two kinds of data processors, the matching and the differencing engine, respectively. These engines are pluggable components: generic engines are provided to match and analyse any model conforming to an arbitrary meta-model (they will be described in the next section) but one can plug new ones in order to specialize these operations for a given meta-model or experiment with new algorithms.

Another strong aspect of this implementation is that we do think models should be leveraged everywhere as already advocated, that's why EMF Compare is based on model representations of both differencing and match of two models. That means one can get those models and use them to produce differences reports thanks to model to text transformation, or can refactor the differencing model to ignore some differences. In this respect, the method is *model-based* according to the requirements in Sect. 2.

3.1. Calculation Method

Analysing models to get the matching information is the fundamental part of the comparison process, inaccuracies in this phase will affect the quality of the overall difference detection mechanism; consequently this algorithm brings most of the calculation complexity. In essence, we have to consider all the elements of both versions of the model and decide whether an element in the first version is the same as another one in the second version. We do use the "same" word as we do not want to test equality, we are just trying to find out if this element has a common ancestor. Next we will check what are the intrinsic differences of these elements to produce the diff model.

The generic match engine is based on statistics, heuristics, and instances are compared with four different metrics aggregated in an overall score of matching. These metrics analyses the name of an element, its content, its type and the relations it has with other elements; it returns a value ranging from 0 (nothing in common) to 1 (identity) which will be balanced with additional factors in order to get the overall score. Especially, the "name" metrics tries to find an attribute standing for a name of the model element, the "type" metric compares the meta-class features, this is useful if you want to consider the possible types refactoring (an `Interface` changed in `Class` for instance). The "relation" metric considers the linked instances both from containment and from non-contained relations, respectively. Finally, the "content" metric analyses the intrinsic content of the instance.

In general, the comparison uses a great deal of information which are not relevant and that can be, therefore, called *information noise*. The metrics gets "false high scores" because most of the data comes from default values which are shared among instances. These corner cases have been treated by means of a filter, which first browses both models and keep track of the features which "always have the same values in both models", then ignores such features while computing the metrics. As a consequence, the metric scores are more realistic as they are not affected by this noise anymore.

We only described the 2-ways comparisons, since the 3-ways one can be given in terms of 2-ways comparisons as specified by the match model. In particular, a difference can be an instance of either `Match2Elements` or `Match3Elements` metaclasses with the latter defined in terms of the several instances of the former. Moreover, model elements which do not have a match are referenced as an `UnMatched` entity.

In order to evaluate the score of a content match, we first create a string representation of what is contained in the instances, and then we compare both strings using a simple "string pairing" algorithm. Each metric uses the same kind of process, first gets a string representation of what we call the "relations" of an element, and then compares these strings.

Ideally, for each element of the first version we have to discover the most similar element of the second version. Unfortunately, most of the complexity lies obviously there as it requires to browse the second model for each element of the first model. In the EMF Compare implementation we started from the following assumptions: most of the things do not change and the probability of moving an element outside its “neighborhood” is really low. Thus, the chosen matching strategy browses both models at the same time, matching the elements available within the limits of a given search window. At the end, elements which are not matched while browsing the models will be compared with each other to produce new matches. The outcome is a match model which is, in turn, passed to the differencing engine which operates in a quite straightforward fashion. In fact, once elements from both models are put in correspondence, they are checked whether the same and eventual difference are evaluated.

With respect to the discussion in Sect. 2, the computation algorithm is based on a measure of similarity and does not fall within the class of methods which make use of persistent identifiers, which makes the computation quite general and suitable for tool chaining and integration. Especially, the decomposition of the algorithm in a matching and differencing module permits the individual reuse of such components disclosing the opportunity to be easily adopted in the realization of additional functionalities, as *model patching*, for instance.

3.2. Representation

Both match and differencing information are represented by means of models which can be reused in model transformations; such models conform to the *match* and *diff* Ecore meta-models [EC08]. A match model is a specialization of a weaving model [B05] which provides associations between elements from the first model to elements from the second one. Another information we encode in the model is the overall score evaluated while performing the matching.

In the rest of the section, the representation mechanism of EMF Compare is evaluated with respect to the properties given in the previous Section 2. In particular, the approach satisfies the *model-based* requirement since the calculated differences are represented through models that conform to the provided *diff* metamodel mentioned above. Being more precise, a difference model reflects the changes made on the first model to obtain the second one, representing them by means of meta-classes like `AddModelElement` or `UpdateAttribute` and difference containers called `DiffGroup`. The `AddModelElement` metaclass has two references, one to the element which has been added in the final model, and another to the corresponding container in the initial one. In this sense, the approach is *metamodel independent* in fact the *diff* metamodel provides constructs able to represent differences between arbitrary models and it does not make any assumption about the metamodel the models being differenced have to be conformant to.

With respect to the *compactness*, the approach produces difference models which represent only the elements involved in the changes and the differenced models are not duplicated as in case of coloring. Moreover, EMF Compare provides facilities to reduce the verbosity and the complexity of the difference models. In order to understand them, let’s consider a rich metamodel like UML2: this metamodel provides a huge expressiveness since each metaclass has a lot of attributes and references. For instance, an association between two classes involves many metamodel elements like `AssociationEnds`, `Properties` and so on. This means that when we compare two UML models, the user is overwhelmed by too many details and analysing them is quite difficult; for instance, an added property may come from the fact that the developer added an association and that it is one of its property ends. To cope with these problems, EMF Compare enables the specification of higher level differences. In particular, by means of meta-model extensions one can contribute a new kind of difference, for instance `AddNavigableAssociation` which will hide the three `AddModelElement` detected for the association and the two properties. With this new kind of difference a new processor is contributed which will refactor the original diff model in order to create the new `AddNavigableAssociation` instances. This is useful to get different kind of granularity on the difference and to handle specific merging in which order matters.

The representation of the differences produced by means of EMF Compare are *transformative* but with some limitations. In particular, each difference model induces a transformation that applied to the first model generates the final one. However, the representation is not context independent since the induced transformation cannot be applied to arbitrary input models but only to the first one used for the difference calculation. Nevertheless, this aspect does not compromise *compositionality* and difference models of subsequent versions of a model can be composed together.

Even if the difference model is deducted from the match one, we do not want it to depend on the match model, that means every information relevant to the difference, and as such needed to merge these differences, should be available in the difference model. This confers to the technique the important *self-containment* property.

3.3 Performance

Manipulating realistic scale models and, in particular, calculating differences between models can pose severe questions about computational efficiency. In fact, performance has been one of the key concerns w.r.t. the generic engines provided with EMF Compare: for instance, the latest release compares two UML2 models of approximately five thousands elements in a few second. Of course, many parameters affect the performances of the comparison, the first one being the number of differences. The more differences we have (especially added and removed elements), the more we need to iterate over the remaining items at the end of the matching process. Model structure is also an important parameter affecting the approach. In fact, a more structured model allows faster comparisons since the structure eases the task of finding matching elements.

This leads to another issue linked with the way the generic match engine browses the models. An instance identity can be often regarded as valid within a certain *locality,* as for instance a package containing a class is definitely an important element for the class identity, but for some other kind of models this assumption does not hold and the browsing strategy is then inefficient.

Finally, the biggest issue with the current implementation is common to many systems based on threshold values, since these thresholds are based on massive experiments on many real world models and are not based on any formal theory nor able to auto-adapt themselves. Though this pragmatic approach is useful and gives encouraging results, it would probably benefit from techniques that prevent elements from being "just under the metric threshold" leading to an inaccurate comparison.

4. Conclusions and Future Work

Model differencing has been intensively investigated over the last few years. There have been some work (e.g., [OWK03, AP03, XS05]) that proposed automated UML–aware differencing algorithms which, in contrast with traditional lexical approaches, such as GNU diff-like tools (see [EGK01, ESS92, FPG03] among others), are capable of capturing the high-level logical/structural changes of a software system. More recently, another approach [LGF07] based on structural similarity extended differencing to metamodel independency, i.e., to models conformant to an arbitrary metamodel. However, the capability of tools to operate on change documentation which conforms only to their own internal format tends to lock software development into a single tool compromising its exploitation as part of a tool chain.

In this paper we presented EMF Compare, a metamodel-independent approach to model differencing based on similarity techniques and fully implemented on the generic modeling platform provided by Eclipse. The problem of model differences presents several difficulties both in the calculation and in the representation. As opposed to other approaches, EMF Compare rigorously adheres to the requirements prescribed in [CDP07] which assure the method to be fully integrable in tool chains where differences can be manipulated or analysed by means of standard model-driven tools. With respect to the work in [LZG04] EMF Compare share many characteristics but provide a clear distinction among representation and visualization whose dividing line is somewhat blurred in the other approach.

Future work include the enhancement of the *transformability* property. In essence, difference models can be viewed as model patches with a certain degree of fuzziness or adjustability in their application. To this end, different models as computed by EMF Compare require to be further transformed in another models conforming to the metamodels introduced in [CDP07], this would essentially need to flat the weaving model given in the difference model as presented here.

Acknowledgements. We would like to thank Antonio Cicchetti and Davide Di Ruscio for the long and insightful discussions on this project.

5. References

[AP03] M. Analen, I. Porres. Difference and union of models. In UML 2003 - The Unified Modeling Language (2003), vol. 2863 of LNCS, Springer-Verlag, pp. 2–17.

[B05] J. Bézivin. On the Unification Power of Models. Journal on Software and System Modeling, 4(2):171–188, 2005.

[BJR04] J. Bézivin, F. Jouault, P. Rosenthal, P. Valduriez. Modeling in the Large and Modeling in the Small. In Model Driven Architecture, European MDA Workshops: Foundations and Applications (2004), vol. 3599 of LNCS, Springer, pp. 33–46.

[CDP07] A. Cicchetti, D. Di Ruscio and A. Pierantonio, A Metamodel Independent Approach to Difference Representation. Journal of Object Technology, vol. 6, no. 9, Special Issue: TOOLS EUROPE 2007, Zurich (Switzerland), October 2007, pages 165–185.

[CW98] R. Conradi and B. Westfechtel. Version models for software configuration management. ACM Computing Surveys, 30(2):232–282, 1998

[EC08] EMF Compare website, http://wiki.eclipse.org/index.php/EMF_Compare, 2008

[EGK01] S. G. Eick, T. L. Graves, A. F. Karr, J. S.Marron, and A.Mockus. Does code decay? assessing the evidence from change management data. *IEEE Trans. Software Eng.*, 27(1):1–12, 2001.

[ESS92] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Trans. Software Eng.*, 18(11):957–968, 1992.

[FPG03] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Procs. ICSM 2003*, pages 23–32. IEEE Computer Society.

[KPP06] Kolovos, D. S.; Paige, R. F. & Polack, F. A. Model comparison: a foundation for model composition and model transformation testing Proceedings of the Int. Workshop GaMMa '06, ACM Press, 2006, 13-20

[In08] Intalio website, http://www.intalio.com/, 2008

[LGF07] Y. Lin, J. Gray, F. Jouault. DSMDiff: A Differentiation Tool for Domain-Specific Models, European Journal of Information Systems (2007) 16, pp. 349–361.

[LMB01] A. Ledeczi, M.Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The Generic Modeling Environment. In Workshop on Intelligent Signal Processing, 2001.

[LZG04] Lin, Y.; Zhang, J. & Gray, J. Model Comparison: A Key Challenge for Transformation Testing and Version Control in Model Driven Software Development OOPSLA Workshop on Best Practices for Model-Driven Software Development, 2004

[M02] T. Mens. A state-of-the-art survey on software merging. IEEE Trans. Softw. Eng. 28, 5 (2002), 449–462.

[Ob08] Obeo website, http://www.obeo.fr/, 2008

[OWK03] D. Ohst, M. Welle, U. Kelter. Differences between versions of uml diagrams. In ESEC/FSE-11: Proc. ESEC/FSE (2003), ACM Press, pp. 227–236.

[S03] B. Selic. The Pragmatics of Model-driven Development. IEEE Software, 20(5):19–25, 2003

[XS05] Z. Xing, E. Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In 20th IEEE/ACM ASE (2005), ACM, pp. 54–65.