A Benchmark for Conflict Detection Components of Model Versioning Systems

Philip Langer and Manuel Wimmer

Business Informatics Group
Institute of Software Technology and Interactive Systems
Vienna University of Technology
{langer,wimmer}@big.tuwien.ac.at

Abstract

In the past years, the model-based development paradigm enjoyed an ever increasing adoption in academia and industry projects. This increasing adoption lead to several diverse approaches of so-called model versioning systems that support collaborative model-based development: multiple developers apply concurrent changes to potentially overlapping parts of the same model and model versioning systems merge their changes to obtain a new consolidated version of the model.

One of the most crucial components in such model versioning systems is responsible for detecting conflicts among concurrent changes to avoid the loss of changes and corrupted models. However, due to the diversity of existing conflict detection components, their relative quality in terms of functional and non-functional properties is hard to determine. To address this issue, we propose a benchmark enabling the automatic evaluation of their accuracy, as well as their execution time, whereas both state-based approaches and operation-based approaches are supported alike.

1 Introduction

Model-based development (MBE) has become a well-adopted development paradigm during the past years. When applying MBE in real-world projects, the complexity and size of the systems being modeled require multiple developers to team up and collaborate on a common set of models. Therefore, dedicated model versioning systems have been proposed that manage concurrent evolution of shared models. In particular, such model versioning systems enable multiple developers to apply changes to the same model in parallel and merge those parallel changes to obtain one consolidated model that incorporates, in the best case, all concurrently applied changes. In many situations, however, concurrent changes cannot be merged automatically as they are in conflict.

In general, a *conflict* in model versioning denotes two changes that either do not commute or disable the applicability of each other. Two changes do not commute, for instance, if they update the same attribute value in the same model element differently; thus, a unique merged model cannot be determined automatically, as the order in which the changes are applied affects the resulting merged model. Two commutative changes may still be in conflict if one disables the applicability of the other: for instance, when one developer deletes a model element that has been updated by another developer, the update operation cannot be applied after the delete and is not reflected in the merged model. The term *conflict* has also been defined formally and more thoroughly, for instance, by Taentzer *et al.* [10] and by Westfechtel [11].

Fortunately, current definitions of conflicts in model versioning seem to converge more and more among researchers and the number of available conflict detection tools is increasing. Nevertheless, comparing these tools regarding their relative quality in terms of functional and non-functional properties is currently a difficult manual task, because an automatic benchmark is missing. Providing such a benchmark, however, is quite a challenge, as these tools use different approaches to obtain the changes, to represent those changes, and to report conflicts.

In this paper, we aim at addressing this challenge by proposing a *synthetic benchmark*[1] that allows to evaluate automatically the *accuracy*, as well as the *execution time*, of conflict detection components in model versioning systems. To make this benchmark applicable to as many tools as possible, we focus in a first step on atomic conflicts (i.e., conflicts among atomic changes, such as add, delete, update, and move) applied to EMF-based models [9], because most of the existing tools implement atomic conflict detection mechanisms and support EMF directly or indirectly (via adapters). Moreover, the benchmark is designed to support model versioning tools that either rely on model comparison or on operation recording.

[1]The benchmark is available at `http://code.google.com/a/eclipselabs.org/p/model-versioning-benchmarks`

2 State-based and Operation-based Change Detection

Before conflicts among changes can be revealed, the changes between two versions of a model have to be identified. Basically, the applied changes can be obtained using two approaches: *operation recording*, which is often referred to as operation-based versioning, and *model differencing*, which is also referred to as state-based versioning (cf. [1] for a survey).

Operation recording depends on the interfaces of the modeling editor or the change notification mechanism of the underlying modeling framework and, basically, records all notifications in order to build a log of changes. In contrast, model differencing is usually realized by, first, computing a *match* between the two model versions under consideration and, second, by identifying the actual *differences* between the pairs of previously matched model elements. Of course, the quality of the match significantly affects the quality of the obtained differences and, in further consequence, also the quality of the detected conflicts. As the benchmark proposed in this paper focuses on the conflict detection, rather than matching, we also provide IDs that are attached to all model elements to allow model differencing approaches to obtain a "perfect" match by exploiting those unique IDs [8].

Both approaches have their advantages: model differencing approaches are independent of the used modeling editor, however, matching and differencing is a computationally expensive task and may not be as precise as operation tracking. Anyway, both approaches are employed in existing tools. Thus, we designed our benchmark to support both of these approaches by providing model versioning scenarios in terms of model states, but also automatically simulate the application of the changes as if they are performed by a user to enable operation-based approaches to record change histories in their specific format.

3 Goal of the Benchmark

The goal of the benchmark proposed in this paper is to enable the evaluation of the accuracy and the execution time of as many different conflict detection tools as possible. Therefore, we use EMF-based models as a common base and focus on generic atomic conflicts (i.e., conflicts that are not related to any language-specific information and that occur between two atomic changes, such as add, delete, update, and move) as a first step, because we are not aware of any other tool than the AMOR conflict detection[2] [7] that is capable of detecting conflicts caused by violations of the preconditions and postconditions of composite operations. Also language-specific conflict detection mechanisms (e.g., by Gerth *et al.* [4] and Langer *et al.* [8]) are quite rare and would require one common modeling language that is supported specifically by all compared tools. Moreover, the benchmark shall support conflict detection tools that build upon state-based model differencing or operation recording, and should cover a wide range of different atomic operation conflict types.

[2]`http://code.google.com/a/eclipselabs.org/p/amor-conflict-detection`

In particular, with the proposed benchmark, we aim at assessing the following questions about conflict detection components in model versioning systems.

1. *Correctness*: Are the detected conflicts correct, or are there scenarios, in which a conflict detection tool raises incorrect conflicts? This property is also referred to as *precision*.

2. *Completeness*: Are the detected conflicts complete or does a conflict detection tool miss to detect any conflicts? This property is also referred to as *recall*.

3. *Execution Time*: What is the execution time for a wide range of different model versioning scenarios? The execution time of the change detection and conflict detection shall be measured separately.

4 Design of the Benchmark

From our evaluations in the past [2], which we performed manually by reconstructing a catalog of versioning scenarios in each tool and documenting the results, we learned that comparing different conflict detection tools is a difficult and tedious task. First of all, different tools require different information; for instance, operation-based conflict detection tools require the information on applied operations in a format they can process and state-based conflict detection tools need the states of a model. Second, the outcome of the tools is heterogeneous and, therefore, difficult to interpret, because each tool reports the conflicts in different ways using different terms and user interfaces. Moreover, performing such an evaluation manually requires a great deal of time, is error-prone, and hardly reproducible. Moreover, whenever a new release of a conflict detection tool emerges, the manual evaluation would have to be repeated.

To avoid the tedious manual evaluation, we developed a *generic application interface* for conflict detection tools and designed an *automatic* benchmark that interacts with this generic interface. In order to evaluate a specific tool, all one has to do is to implement the generic interface for the specific tool to be evaluated and start the benchmark.

In the following, we provide details on the the generic interface for conflict detection tools, the versioning scenarios that are used to evaluate conflict detection tools, as well as the measures computed from the obtained results.

Generic conflict detection API. To allow our benchmark to uniformly work with all conflict detection tools to be evaluated, we developed a generic conflict detection API, which is depicted in Figure 1. This tool-independent programming interface incorporates the types and methods required to initiate the detection of conflicts for a specific versioning scenario irrespectively of whether the evaluated conflict detection tool builds upon state-based model differencing or operation recording in the editor. More specifically, the API contains the interface IBenchmarkableFacade, which represents the exterior simplified interface to the conflict detection tool. This facade dictates several methods to be implemented. These methods are called by the benchmark to provide the conflict detection tool with all the required information for detecting conflicts for a specific model versioning scenario. This information is, on the one hand, the original model resource, as well as the left and right revised model resource. To also allow conflict detection tools that are based on operation recording to collect the applied operations on each side, the benchmark further sets the left and right editing domain in which the operations of the respective model versioning scenario are automatically applied by the benchmark. Thereby, operation tracking approaches may exploit the command API of EMF to track every applied operation as if a user would perform the respective operations. In case, operation-based approaches rather use the notification framework for tracking operations, they may simply register themselves as observer to the specified origin model. Once, all operations of the left side and right side have been automatically applied, the benchmark provides the revised left and right model to the conflict detection facade and calls the method getConflicts(). This method should return a list of all detected conflicts for the particular model versioning scenario in the form of instances of the IConflict interface. Such an instance of this interface describes the involved model elements, as well as the respective conflict type. Having the information on the detected conflicts, the benchmark compares the detected conflicts provided by the evaluated tool with the expected set of conflicts.

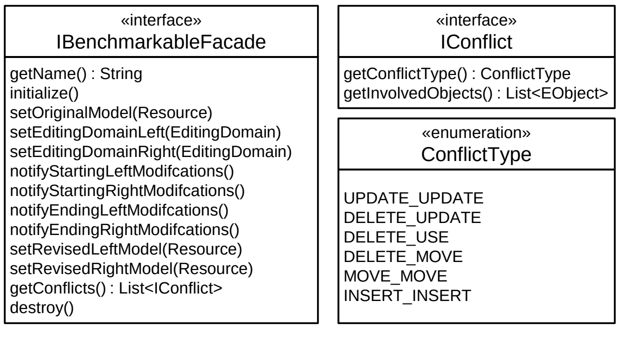
«interface»
IBenchmarkableFacade
getName() : String
initialize()
setOriginalModel(Resource)
setEditingDomainLeft(EditingDomain)
setEditingDomainRight(EditingDomain)
notifyStartingLeftModifcations()
notifyStartingRightModifcations()
notifyEndingLeftModifcations()
notifyEndingRightModifcations()
setRevisedLeftModel(Resource)
setRevisedRightModel(Resource)
getConflicts() : List<IConflict>
destroy()
«interface»
IConflict
getConflictType() : ConflictType
getInvolvedObjects() : List<EObject>
«enumeration»
ConflictType
UPDATE_UPDATE
DELETE_UPDATE
DELETE_USE
DELETE_MOVE
MOVE_MOVE
INSERT_INSERT

Figure 1: Generic Conflict Detection API

Implementing the interfaces. For evaluating conflict detection tools, the benchmark executes each scenario by first initializing the conflict detection tools, setting the origin model as well as the editing domains and then replaying the operation sequences to the origin model within the editing domains. Finally, it provides the left and right revised models, obtains the detected conflicts of each tool, and computes the evaluation results in terms of specific measures for each evaluated tool. More information on how to implement the interfaces and how the benchmark calls the implementation of a conflict detection component is provided at a dedicated Wiki page [6].

Versioning scenarios. The benchmark consists of 23 purposefully selected versioning scenarios, which are partly taken from the collaborative conflict lexicon [2], we developed to induce a broad collection of conflict scenarios, and partly from our experiences gained from practice. These scenarios cover different types of atomic conflicts (delete/update, update/update, move/move, delete/move; cf. [10]). Furthermore, these versioning scenarios also include cases in which no conflict is expected at all. Although we believe that these scenarios cover the most common cases concerning the detection of conflicts, the benchmark should be further enhanced in future by the model versioning community. Therefore, we implemented a dedicated mechanism that allows to add additional versioning scenarios easily by recording two change sequences and denoting the expected conflicts.

Besides dedicated versioning scenarios for evaluating the accuracy of conflict detection components, the benchmark also contains performance tests in terms of small to large models, as well as small to large change histories that are applied to these models. These scenarios are executed repeatedly with integrated conflict detection tools and the average execution time of detecting the conflicts is measured.

Measures. To assess the accuracy of the considered conflict detection tools, we compute the measures *precision* and *recall*. In the context of conflict detection, we define the precision as the fraction of *correctly detected* conflicts among the set of *all detected* conflicts (i.e., how many detected conflicts in fact are correct). The recall indicates the fraction of *correctly detected* conflicts among the set of *all actually occurred* conflicts (i.e., how many conflicts have not been missed). As these two measures may be thought of as probabilities, their values may range from 0 to 1. We further compute the *accuracy*, which is obtained from the number of correctly raised conflicts (*tp*), number of correctly handled conflict-free scenarios (*tn*), number of incorrectly raised conflicts (*fp*), and the number of missed conflicts (*fn*) as follows: $\text{accuracy} = \frac{\text{tp}+\text{tn}}{\text{tp}+\text{tn}+\text{fp}+\text{fn}}$

For evaluating the performance efficiency of conflict detection components, we measure the average execution time of multiple executions in *milliseconds*, whereas the time required for change detection and conflict detection is recorded separately.

5 Conclusion

We believe that a generic benchmark for conflict detection components provides a valuable asset not only for determining their relative quality but also for developers of conflict detection components, as such a benchmark documents requirements in a systematic and verifiable manner. To gain first experiences and results from the benchmark, we already implemented the interface of the benchmark for the AMOR conflict detection component [7], EMF Compare [3], and EMFStore [5]. Having created these interface implementations, we applied the benchmark successfully to all three considered tools. The results of this evaluation is presented in [7, Section 7.3.3].

When interpreting the bechnmark results, it is important to put the computed measures in relation to the general approach that is applied in the conflict detection components (e.g., state-based versus operation-based versioning). It is obvious that accomplishing fast and accurate components using operation-based change recording is much easier than realizing fast and accurate components that apply state-based model differencing; especially when they do not rely on unique identifiers in the matching phase of the model differencing process. Thus, we propose to evaluate these diverse approaches in distinct categories: *(i)* operation recording approaches, *(ii)* identifier-based model differencing approaches, and *(iii)* similarity-based model differencing approaches.

Interesting lines of future work comprise the establishment of a dedicated *versioning scenario modeling language* with the aim to provide an easy and declarative way of specifying versioning scenarios consisting of an initial model, two concurrent change sequences, and the list of conflicts that are expected. Moreover, we plan to extend the currently existing set of versioning scenarios. Therefore, we kindly invite the model versioning community to contribute and help us working towards establishing a commonly accepted and thorough model versioning benchmark.

References

[1] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer. An introduction to model versioning. In Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio, editors, *Formal Methods for Model-Driven Engineering (FSE'12)*, volume 7320 of *LNCS*, pages 336–398. Springer, 2012.

[2] P. Brosch, P. Langer, M. Seidl, K. Wieland, and M. Wimmer. Colex: A Web-based Collaborative Conflict Lexicon. In *Proceedings of the International Workshop on Model Comparison in Practice @ TOOLS'10*, pages 42–49. ACM, 2010.

[3] C. Brun and A. Pierantonio. Model Differences in the Eclipse Modeling Framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29–34, 2008.

[4] C. Gerth, J. M. Küster, M. Luckey, and G. Engels. Precise Detection of Conflicting Change Operations Using Process Model Terms. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, volume 6395 of *LNCS*, pages 93–107. Springer, 2010.

[5] M. Koegel, M. Herrmannsdoerfer, O. Wesendonk, and J. Helming. Operation-based Conflict Detection on Models. In *Proceedings of the International Workshop on Model Comparison in Practice @ TOOLS'10*, pages 21–30. ACM, 2010.

[6] P. Langer. Implementing the Conflict Detection API. `http://code.google.com/a/eclipselabs.org/p/model-versioning-benchmarks/wiki/ImplementingIBenchmarkableFacade`, retrieved in May 2013.

[7] P. Langer. *Adaptable Model Versioning based on Model Transformation by Demonstration.* PhD thesis, Vienna University of Technology, 2011.

[8] P. Langer, M. Wimmer, J. Gray, G. Kappel, and A. Vallecillo. Language-Specific Model Versioning Based on Signifiers. *Journal of Object Technology*, 11(3):1–34, 2012.

[9] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *Eclipse Modeling Framework 2.0.* Addison-Wesley Professional, 2008.

[10] G. Taentzer, C. Ermel, P. Langer, and M. Wimmer. A fundamental approach to model versioning based on graph modifications: from theory to implementation. *Software and Systems Modeling*, 2012.

[11] B. Westfechtel. Merging of EMF models - formal foundations. *Software and Systems Modeling*, 2012.