The Case for Batch Merge of Models – Issues and Challenges

Lars Bendix[1], Maximilian Koegel[2], Antonio Martini[3]

[1] Department of Computer Science, Lund University, S-221 00 Lund, Sweden
[2] Institut für Informatik, Technische Universität München, D-85748 Garching, Germany
[3] Dipartimento di Informatica, Università di Parma, I-43100 Parma, Italy

bendix@cs.lth.se, koegel@in.tum.de, anto@ideazan.it

Abstract. Modelling is a fairly mature technology that is already being adopted by industry. Unfortunately the tools for supporting Model-Driven Engineering are not at the same level of maturity. In particular merge tools that are an important help for collaboration lack a lot to be desired in providing the support that is needed on industrial projects. Current model merge tools and research approaches focus on interactive merging where the developer resolves possible merge conflicts one by one during the merge process. However, developers are used to merge tools from the textual domain that work in a batch mode – the merge tool is provided with three files as input and produces a file with the proposed merge result – including possible unresolved merge conflicts. The developer can then pick up this result and resolve possible conflicts at a time and in an order that pleases him. In this paper, we motivate why research should start to focus also on batch merge for models and we describe how it could work. Furthermore, we sketch the research agenda that is needed to address the issues and challenges in realizing batch merge for models.

Keywords: Model merge, interactive vs. batch mode, conflict representation, conflict management, issues and challenges, research agenda.

1 Introduction

In a software development project, models are important artefacts and they have become more and more important in recent years, especially with the adoption of Model-Driven Engineering. Working with and evolving models in an industrial context is a highly collaborative activity where multiple developers participate. Therefore it is important to be able to manage changes to the models and to manage conflicting changes carried out by different developers working in parallel. The activity of integrating such changes into a consolidated model is called merging and involves resolving conflicting changes to build a new resulting model that reflects the intentions of all involved developers as far as possible.

The road that model merging has taken so far differs from the way in which merge works in a traditional textual context. The goal of model merge is to produce the "perfect merge result" where all conflicts are resolved. When the merge tool encounters a conflict that cannot be resolved automatically, it asks the developer to resolve the conflict – usually constraining the choice to one of the two alternatives. The advantage is that new work can continue immediately after the merge; the drawback is that we may force the developer to make a lot of decisions up-front. Traditional textual merge, on the other hand, has no illusions of perfection. They happily ignore syntax and semantics of what is being merged and whenever a conflict is encountered it is just marked in the text. The advantage is that a merge result is produced very quickly without any user intervention; the drawback is that the developer should put the result right (or at least check if it is right) before continuing his work.

We have experienced a great desire from industry to be able to work with model merge in the same flexible way as when working with merge for traditional programming. In the following, we look at the further motivation for batch merge of models, then we analyse what issues and challenges need to be addressed. Finally, we briefly discuss some existing research and draw our conclusions.

2 Motivation

In practical software development it often happens that work is carried out in parallel for many different reasons. Traditionally parallel work is handled by selecting a suitable branching strategy [1] that will support the specific type of parallel development that is carried out. However, whenever we talk about branching out to several lines of development, we also create the "double maintenance problem" [2] which means that changes to one branch should also – sooner or later – be done to the other branch(es) to keep them synchronized. Integrating changes between branches is done using a merge tool and is a very frequent activity.

From a previous study of model merge in industry [4], it turned out that, besides quality issues with the results produced by the merge tools, a major complaint was that the developers found interactive merge an awkward and disruptive way of working. The fact that there might be a long session of resolving conflicts before the merge would be done meant that there was a perceived high mental cost involved in starting a merge, thus discouraging people from doing it as frequent as they might have wanted. Sometimes the right choice was not one of the alternatives, but a mixture, which was not possible. There would also be occasions where they would regret a previous choice later on in the process, and in certain situations they would need to consult another person – that was not there – to resolve the conflict.

This leads us to propose a "new" merge strategy for model merge – batch merge – where the total cost of the merge process can be split up into two "stages" (merge production and conflict resolution) that can be carried out independently and when it is convenient for the developer. Even in the case where conflict resolution is initiated immediately after the merge production, batch merge has advantages. Interactive merge forces developers into resolving conflicts in a particular order – that of the tool. Batch merge would allow for "sequence independent conflict resolution". Batch merge is not "new" as it is the way that textual merge is done and we would like to see something similar for model merge.

We will now express these ideas in a number of more detailed scenarios. These scenarios came up during interviews and discussions with developers and from drawing on our own experience:

Produce a merge – the two alternatives and the common ancestor of the system (often consisting of several files) are fed into the merge production tool. It produces a system where the conflicts are marked and the conflicting alternatives are represented in the model.

Discuss conflicts in asynchronous collaboration – you need to discuss a conflict and collaborate with someone else to figure out how to resolve it. You can either postpone the resolution until that person is present or you can send the merge result to the person and ask him to resolve the conflict.

Resolve conflict – working from a merge result, it is possible to allow the developer to choose the order in which conflicts are resolved. Furthermore, it is possible to take a more global view of the conflicts and discover connected conflicts that have to be resolved as one logical entity.

Virtual merge – shows you what happens if you merge, but does not do the actual merge. You only want to have an overview of where conflicts and changes are (like you can do in the Ragnarok system [5]) to be aware of what is happening and do not want to have to interactively "resolve conflicts" first.

3 Issues and challenges

During our analysis of issues and challenges in providing batch model merge we identified many different topics. For presentation reasons, we have grouped them into two categories – merge production and conflict management – and for lack of space, we state only the most important ones (and leave out those that do not fit into these two categories).

3.1 Merge production

For the production of a merge result there are some things like model matching and conflict detection that are common for both interactive merge and batch merge. However, in batch merge production there is no conflict resolution integrated. The merge production is fully automated and there is no possibility to ask for developer intervention. The consequence of this is that there are two major differences between batch and interactive merge: the produced output might not be a valid model and that we will have to mark and represent conflicts in some way. Besides that there are a number of minor particularities.

Conflict mark-up – in the case where there have been concurrent changes to the same Unit of Comparison (UC) we have a conflict. There may be cases in which a careful analysis will be able to produce a correct result, but in general there is no way to guarantee a correct result. Interactive merge relies on "human intelligence" from the developer intervention to produce a correct result. In batch merge we will have to give up and signal a conflict for the UC in question. This conflict will have to be represented in the produced result in some way to allow the developer to discover that there is a conflict – how that can and should be done is an open question.

Alternative representation – when a conflict is detected it means that the same UC has been changed in two different ways. Besides signalling the conflict in the result we also need to incorporate information about what the two alternatives are. This will help the developer to resolve the conflict later on with the same information available as for interactive merge.

Violation handling – for textual merge the question of when there is a conflict is simple: when there are concurrent changes to the same UC. Since textual merge ignores the possible syntax and semantics of what is merged, it also ignores potential problems with the syntax and semantics of the produced result. This differs completely from the approach in interactive model merge where there are a number of constraints on what is a legal output. This means that interactive model merge talks about hard and soft conflicts – hard conflicts being cases where the result breaks the meta-model constraints; soft conflicts being cases where the result breaks other validation constraints, but still conforms with the meta-model. A merge result with soft conflicts will load in the model editor, whereas a result containing hard conflicts will not. Therefore interactive model merge goes to great length to discover and avoid hard conflicts. For batch merge it is doubtful whether this should be considered part of merge result production since it requires more analysis and work than just "simple" conflict detection at the level of the same UC. Furthermore, batch merge already has to handle merge results that might not conform to the meta-model, so this might be considered just one more thing that should be taken into consideration when designing the meta-model for the merge result.

Change mark-up – when a UC is changed by only one of the parties, it should be straightforward to include the changed UC in the merge result. However, as it was pointed out above, that might lead to cases where the merge result violates some constraint imposed either by the meta-model or in other ways. Such violations will have to be discovered and sorted out in the subsequent conflict management phase. Furthermore, information about other changes may be useful when the developer has to resolve real merge conflicts. For these reasons we suggest to consider the mark-up also of "simple" changes to avoid loss of information from the merge production phase to the conflict management phase. This information could be obtained from a diff functionality, but it seems better to just put it in the result right away.

Merge result format – interactive model merge can count on developer intervention for sorting out conflicts and can thus produce a merge result that conforms to the meta-model and is ready to use. For batch merge we do not have that possibility. We will have unresolved conflicts that need to be marked up in the result. We will have to represent the alternatives in the case of a conflict. If we chose to ignore validation constraints in the merge production phase, we will have to deal with merge results that might not conform to the meta-model for what we are trying to merge. Finally, we would like to mark up all changes (also those that do not create conflicts) to avoid loss of information. This means that we have to come up with a format or model merge meta-model that is sufficiently robust and complete to be able to represent all these possibilities for the merge result. Then the subsequent conflict management phase can be used to gradually bring the merge result in a state that it conforms to the original meta-model – and possible additional validation constraints. How such a model merge meta-model should look like and what are the requirements (in particular for the violation handling aspects) is a widely open question.

Merge input format – since the merge production phase creates a result that may not respect the meta-model, we have to consider what should happen if a developer uses the result as the input to a new merge without – or before – bringing it in conformance with the meta-model. The ideal solution would be a merge production that is able to handle not just input that conforms to the meta-model, but also inputs that break the meta-model but still conform to the model merge meta-model. The advantages of such a solution are clear, however, the difficulties in implementing such a merge are far from clear.

3.2 Conflict management

The result that is created by batch merge is not guaranteed to be correct or work – rather it is almost guaranteed to be inconsistent in one or more ways with respect to the developer's intentions and expectations. In the second phase of the merge process – conflict management – the developer can work on the produced merge result to bring it in a state that is satisfactory for him and reflects his intentions. He needs to inspect and change the merge result, to resolve conflicts and other problems with the help of functionality that can analyse the merge result, and to continuously validate the result to discover remaining problems.

Visualization and editing – we will need new tools to visualize and edit the merge result produced by batch merge since it will probably not conform to the meta-model. However, when a standard model merge meta-model is established, it should be no problem to build such tools. Current model editors ignore visualization aspects of model merge – probably because there are no such aspects to visualize when interactive merge is used. However, as it was discussed above for *virtual merge*, visualization can be of great help for creating awareness about how the system changes.

Model validation – since the merge result does not conform to the meta-model, it is very important to be able to validate the resulting model at any given time. The validation process should be simple and quick, and the validation tool should be able to work with the model merge meta-model and point out where and what does not conform to the original meta-model.

Connected conflicts discovery – some conflicts (and/or changes) may be connected/related. The prime example is a refactoring. In batch merge all changes and conflicts are directly represented in the result and an intelligent tool can analyse the result to find connected/related conflicts/changes so they can be presented – and resolved – together. A special case of this is for operation-based merge, where conflicts have options, options can be conflicting with the options of other conflicts or they can require a certain option of another conflict.

Rich alternative proposal – since we do not have to respect the original meta-model – only the model merge meta-model – we can propose more options for conflicts resolution than just “left or right alternative”. We can even allow the developer to edit the proposal immediately.

4 Conclusions

In this paper, we have motivated why batch merge for models is not purely "an academic exercise", and demonstrated several situations where batch merge will be of great practical use in an industrial context. We have identified a number of issues and challenges that will have to be addressed before batch merge can become a reality.

Part of the research agenda for batch merge should be revisiting existing research results from interactive model merge. Issues like operation-based [7] and state-based [10] merge will probably have to be reconsidered, just like the trade-off for Unit of Versioning and Unit of Comparison [9] will most likely change. A new model merge meta-model will be the fundamental interface between the merge engine and the merge management tools. There exist some initial ideas from various contexts [6], [3], [8] that can be used as a starting point, but there is still a long way to go.

A side effect of batch merge is a very nice and useful separation of concerns – and tools – both for industry and for research. Industry can use one tool for "merge production" and another tool(s) for "merge management" – research can proceed in parallel in both areas and will not have to do all-inclusive implementations just to try out some new research idea.

References

1. Appleton, B., Berczuk, S. B., Cabrera, R., Orenstein, R., *Streamed Lines – Branching Patterns for Parallel Software Development*, in proceedings of the Pattern Languages of Programming Conference, Monticello, Illinois, August 11-14, 1998.
2. Babich, W. A., *Software Configuration Management – Coordination for Team Productivity*, Addison-Wesley, 1986.
3. Bartelt, C., *Consistence Preserving Model Merge in Collaborative Development Processes*, in proceedings of the International Workshop on Comparison and Versioning of Software Models, Leipzig, Germany, May 17, 2008.
4. Bendix, L., Emanuelsson, P., *Requirements for Practical Model Merge – an Industrial Perspective*, in proceedings of the 12th International Conference on Model Driven Engineering, Languages and Systems – MODELS '09, Denver, Colorado, October 4-9, 2009.
5. Christensen, H. B., *Tracking Change in Rapid and eXtreme Development: A Challenge to SCM-tools?*, in proceedings of the Tenth International Workshop on Software Configuration Management, Toronto, Canada, May 14-15, 2001.
6. Cicchetti, A., Di Ruscio, D., Pierantonio, A., *A Metamodel Independent Approach to Difference Representation*, Journal of Object Technology, Vol. 6, No. 9, 2007.
7. Kögel, M., Hermannsdoerfer, M., von Wesendonk, O., Helming, J., *Operation-based Conflict Detection on Models*, in proceedings of the International Workshop on Model Comparison in Practice, Malaga, Spain, July 1, 2010.
8. Martini, A., *Merge of models – an XMI approach*, Master's Thesis, Department of Computer Science, Lund University, August, 2010.
9. Oliviera, H., Murta, L., Werner, C., *Odyssey-VCS: A Flexible Version Control System for UML Model Elements*, in proceedings of the 12th International Workshop on Software Configuration Management, Lisbon, Portugal, September 5-6, 2005.
10. Westfechtel, B., *A Formal Approach to Three-Way Merging of EMF Models*, in proceedings of the Workshop on Model Comparison in Practice, Malaga, Spain, July 1, 2010.