Collaborative Modelling with Version Control

Steven Kelly http://orcid.org/0000-0003-0931-157X

MetaCase, Jyväskylä, Finland
stevek@metacase.com

Abstract. Modelling and version control both play key roles in industrial-scale software development, yet their integration has proved difficult. Significant effort has been expended on improving file-based merging of modellers' work, but empirical research shows the results of that approach still leave much to be desired in practice. Approaches based on multi-user modelling databases have often foundered by locking too broadly, thus preventing work on different elements in the same model, or by handling versioning themselves, leading to a silo. This article presents an approach to combining multi-user modelling with any external version control system, with no merging and no lock-outs.

Keywords. multi-user, modelling, locking, merge, version control systems

1 Introduction

Around 15 years ago, model-driven development faced a crossroads. Where models became primary assets in software development, should collaborative work on them follow the multi-user repository approach of leading modelling tools, or the clone and merge approach that had worked so well on the earlier primary assets, source code files? In academic research, the balance swung towards XMI model files under source code version control systems. In industrial use, both approaches have continued to this day, with multi-user functionality in tools like MagicDraw, Enterprise Architect and MetaEdit+[1]. Both approaches have been fruitful, yet combining them seamlessly to get the best of both worlds has proved elusive, particularly for the increasingly important case of distributed version control systems.

This article presents an approach to combining multi-user modelling in any languages with any external version control system, with no merging and no lock-outs.

2 Background and Related Research

The issue of collaboration and versioning is central to the scalability of model-driven development teams, as noted in two key items in the roadmap laid out in BigMDE 2013 [1]: "support for collaboration" and "locking and conflict management".

[1] nomagic.com/products/magicdraw, sparxsystems.com/products/ea, metacase.com/mep

2.1 Two ways to collaborate

With multiple developers interacting on a large project, it is generally not feasible to divide the system into isolated modules, one for each developer. We thus have to cope with integrating the work of several simultaneous developers on a single module.

There are two different ways to approach the problem of multiple users. The first way gives each user their own copy of the files to be edited and allows any changes, with the first user to version getting a ”free ride”, and others having to merge their changes into that version after the fact. The second way allows users to work on the same set of models, integrated continuously as they save, with locks to prevent conflicting changes.

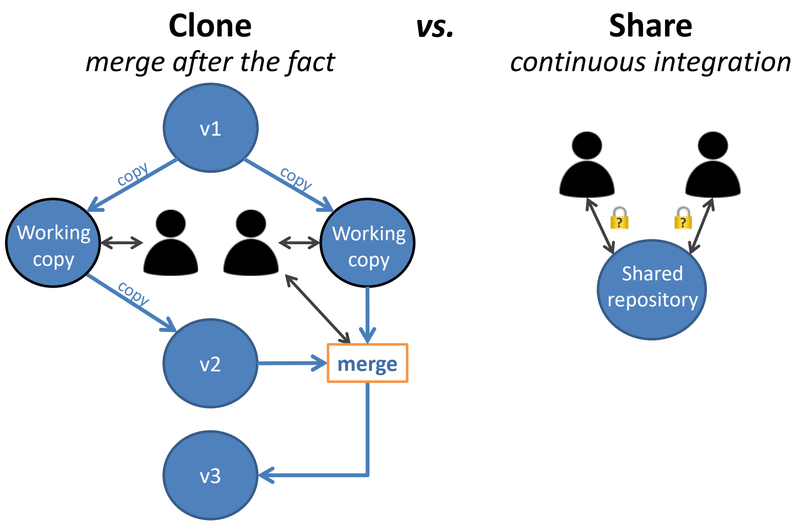


Fig. 1. Two ways to collaborate: Clone vs. Share

Although it is common in the research literature to refer to these two approaches as “optimistic locking” and “pessimistic locking”, these are not particularly accurate terms. In the former there is actually no locking, and in the latter there is no pessimism: a lock is taken only when necessary or requested, not as a worst-case precaution. We will thus refer to them here as “clone and merge” and “share and lock”, describing what actually happens.

2.2 Research on the “clone and merge” approach

The vast majority of research has concentrated on the problem that VCS algorithms for merge are designed for simple text files, and applying them to the more highly structured and interlinked data in model files leads to results that vary between inconvenient and unacceptable. Out of 470 articles in the University of Siegen’s excellent

Bibliography on Comparison and Versioning of Software Models[2], almost all focus on this approach. In Europe, the MODELWARE (2005–2006) and MODELPLEX (2006–2009) projects, each costing around 20 million Euros, included significant effort on this topic. With that scale of research and effort, it seems fair to assume that theory and implementation are already close to the best that is possible on this path, and indeed we make no effort here to offer incremental improvements to them. There is a good overview of the issues in [2] and the current state in [3].

Recent empirical evaluations of user experiences with the “clone and merge” approach show significant problems remain: “Exchanging information among tools using XMI creates many problems due to the format misalignment in the various tools”, “all companies try to avoid merging as much as they can”, differencing “is error-prone and tedious” [4]. Experiences with a major vendor’s UML tool revealed “difficulties in merging… became impossible”, leading to “trying not to branch” [5].

2.3 Research on the “share and lock” approach

Ohst and Kelter found that improving locking in ADAMS to use a finer granularity reduced the need to branch and merge from 30% of cases to zero [6].

Gómez et al. have tried to bring a locking approach to EMF with HBase [7], but the lack of explicit transaction support meant ACID properties were lost above the level of single changes: unacceptable for modelling, where a coherent unit of work requires several changes.

Odyssey-VCS 2 is a version control system tailored to fine-grained UML model elements [8]. It moved away from explicit branching to auto-branching, provided better information of a single user’s changes and intentions, and allowed locking at a fine granularity. However, locks must be taken explicitly, and if unused must be freed explicitly too.

Systemite’s SystemWeaver offers explicit versioning within a database, although their data is more focused on components than graphical models, and they offer no external VCS integration. They criticise file-based approaches, citing three studies showing that 14-30% of engineers’ time is spent on finding the correct data and version [9].

3 Current State of the Art

In this section, we examine a state of the art example of each approach.

3.1 Clone and Merge: EMF Compare + EGit

The file-based approach is easy to explain: model files are treated like source code files, so all the normal operations of Git or similar are available – but also required. The main benefit of EMF Compare [10] is to offer diff and merge algorithms better-

2 http://pi.informatik.uni-siegen.de/CVSM/

suited to EMF XMI models, and to display results in a partially graphical format. The simplest EMF Compare + EGit integration seen to date is in an EclipseSource video[3] by Philip Langer. Even there, versioning with a single simple conflict requires a sequence of 11 user operations: "branch + checkout, save, add + commit, checkout, pull, merge, resolve conflicts, save, add, commit, push". Even without a conflict, the sequence is "branch + checkout, save, add + commit, checkout, pull, merge + commit, push": seven manual operations which the user has to remember and find within the various menus of Eclipse. (Watching the video is recommended to understand the current state of the art, and compare the improvements offered in this paper.)

Is this complexity essential, or could another way show it to be accidental complexity, introduced by applying tools and methods designed originally for simple text files to the rather different world of models? Indeed, could the more structured nature of models actually help us avoid the need for this complexity? There are certainly substantial gains to be found: empirical research finds that modellers spend over one hour a day on interacting with version control systems [11].

3.2 Share and Lock: MetaEdit+

The second way has its origins in the richer structures of databases. In today's multi-user modelling tools, the second way is seen in MetaEdit+ [12] or the collaboration mode of the commercial version of Obeo Designer[4]. The basics of the approach are similar between the tools, but here we will discuss the approach taken in MetaEdit+, as it will also serve as a starting point for our later discussions.

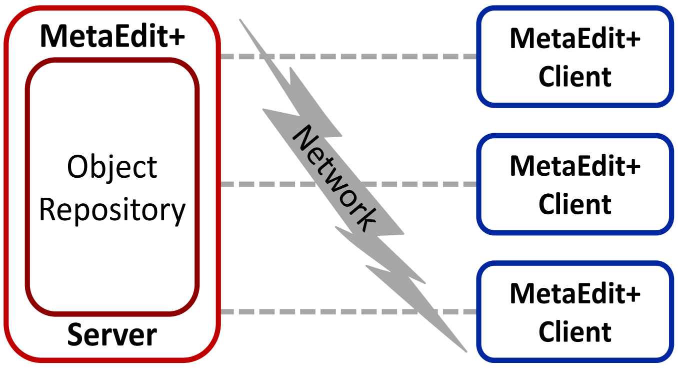


Fig. 2. MetaEdit+ multi-user architecture

Figure 2 shows the architecture of the MetaEdit+ multi-user version [13], which we describe in this section as it was from release in 1996 to before the work in this paper. The heart of the multi-user environment is the Object Repository, a database running on a central server. All information in MetaEdit+ is stored in the Object Repository,

3 https://youtu.be/NSCfYAukYgk
4 https://www.obeodesigner.com/

including modelling languages, diagrams, objects, properties, and even font selections. Clients access the repository over the network via the lightweight MetaEdit+ Server; the user interface and execution of all modelling operations is in the clients.

The Object Repository itself is designed to be mostly invisible to users. Objects are loaded as needed and cached by clients when read, and thus are only read once per session over the network: performance after that initial read is identical to normal objects. Because of the JIT loading of objects, and no need to parse XML, performance of opening a model compares favourably with XML-based tools, particularly for large models [14]. This corresponds to the "Efficient model storage" and "Model indexing" items of the scalability roadmap [1]. Objects are also automatically flushed from the cache if models become larger than the configured desired memory usage, allowing seamless work and global queries on massive models (a similar approach has been used later by Daniel et al. on Neo4EMF [15]).

An extra layer of consistency is familiar from ACID rules for database transactions: during a transaction, users see a consistent view of the repository as it was at the start of their transaction. This gives the ease of Google Docs multi-user editing, without the distraction of others' half-finished changes happening before your eyes – a problem that users of other tools have described as feeling "like shooting at a moving target" [16].

Fine granularity locks ensure no conflicts, while letting users work closely together in parallel. Making a change automatically takes a minimal lock, down to the level of granularity of a single property, preventing conflicts without preventing users working closely together. When a user has finished making a coherent set of changes, he can commit and his changes are then automatically available to other clients, with no need for manual integration.

Whereas the "clone" approach to collaboration gives rise to versions as a welcome side-effect, the "share" approach does not itself create versions. Although VCS functionality is not needed here to enable collaboration, we still need versioning in order to be able to record what we have shipped. To save the state of the repository to a version control system, all the users first had to log out and the server process had to be stopped: an unwelcome interruption and a harsh return to the file-based world.

Similarly, whereas the "clone" approach must calculate differences, and by convention requests the developer to enter a human-readable description of the changes, the "share" approach does not do this. We still need to see what has changed as an aid to documentation, bug hunting and impact analysis, so extra functionality is needed to help developers capture that information.

4 Comparison functionality in a modelling tool

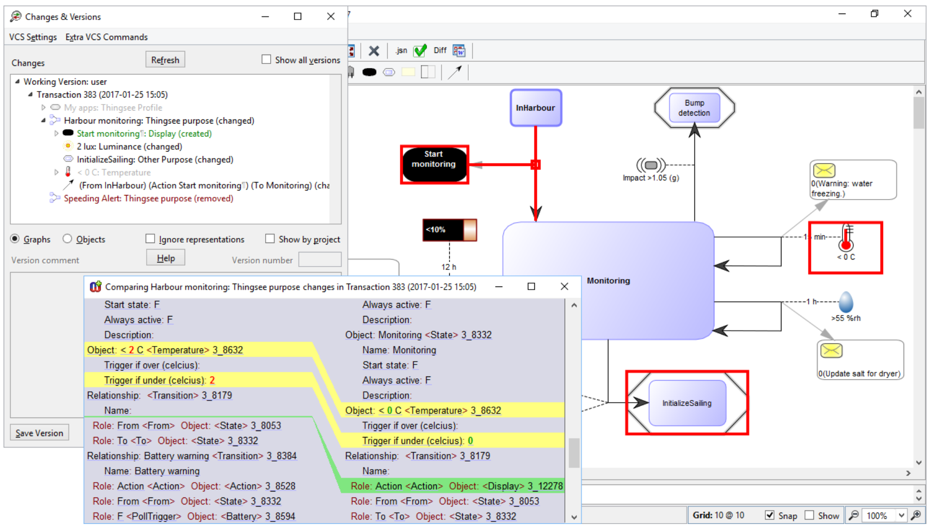
We want a way to bring together the best of both approaches above. We will start from MetaEdit+ as described above, and show in this section how we have now added comparison functionality directly in the modelling tool, avoiding the complication of reconstituting it by comparing lower-level XMI files. The section after this covers the other novel addition of this paper, integration with external version control systems.

4.1 View changes as a tree

The Changes & Versions Tool shows what changes users have made, helping them document the version history of their models. A Version is made by a user on demand, and consists of the one or more transactions by that user since his previous Version. The history is shown as a tree with Versions at the top level, containing that user's Transactions, containing Graphs and Objects that changed in that transaction.

By default the tree shows only the current Working Version of the current user, but 'Show all versions' broadens this to previous versions and all users. Users can choose 'Ignore representations', so simple graphical rearrangements of diagram elements do not clutter the display.

Colour and symbols are used to highlight specific types of change: green for new, red for deleted, black for changed, and grey for elements that are not themselves changed, but which are parents for changed elements.



Working Version: user
  Transaction 383 (2017-01-25 15:05)
    My apps: Thingsee Profile
    Harbour monitoring: Thingsee purpose (changed)
      Start monitoring¶: Display (created)
      2 lux: Luminance (changed)
      InitializeSailing: Other Purpose (changed)
      < 0 C: Temperature
      (From InHarbour) (Action Start monitoring¶) (To Monitoring) (cha
    Speeding Alert: Thingsee purpose (removed)

Comparing Harbour monitoring: Thingsee purpose changes in Transaction 383 (2017-01-25 15:05)

```
    Start state: F                                              Always active: F
    Always active: F                                            Description:
    Description:                                            Object: Monitoring <State> 3_8332
Object: < 2 C <Temperature> 3_8632                              Name: Monitoring
    Trigger if over (celcius):                                  Start state: F
    Trigger if under (celcius): 2                               Always active: F
Relationship:  <Transition> 3_8179                              Description:
    Name:                                                   Object: < 0 C <Temperature> 3_8632
  Role: From <From>  Object: <State> 3_8053                     Trigger if over (celcius):
  Role: To <To>  Object: <State> 3_8332                         Trigger if under (celcius): 0
Relationship: Battery warning <Transition> 3_8384           Relationship:  <Transition> 3_8179
    Name: Battery warning                                       Name:
  Role: Action <Action>  Object: <Action> 3_8528              Role: Action <Action>  Object: <Display> 3_12278
  Role: From <From>  Object: <State> 3_8332                   Role: From <From>  Object: <State> 3_8053
  Role: F <PollTrigger>  Object: <Battery> 3_8594             Role: To <To>  Object: <State> 3_8332
```

Fig. 3. Changes & Versions Tool with tree view, graphical view and text view

4.2 Graphical comparison to highlight changes

For a selected graph version in the Changes tree, 'Open' will open the current diagram and highlight the elements changed in that version. For many cases, this highlighting gives a quick, intuitive, graphical overview of what has changed, shown in the context of the current state of the model. For the full details, e.g. of sub-objects not directly represented in a diagram, users can use the tree or textual Compare.

4.3 Textual comparison with model links

From the Changes tree, users can choose ‘Compare’ to compare the selected Version, Transaction or Graph with its predecessor. This will open a text comparison of the selected state with its previous state. The text elements have Live Code hyperlinks, so double-clicking a link will open that element in its current context in the model. The text lists the graph, its objects and relationships, together with all their details and links to any subgraphs. This approach is intended as a good compromise between coverage and readability. It is produced by a normal MERL generator, and so can be customized where necessary: domain-specific diff.

When comparing a Version, the comparison is to the previous Version, hence multiple graphs across multiple transactions. Rather than show each graph many times, each time with just the changes made in a single transaction, the changes for each graph will be combined and shown from the initial state to the final state. Only if other users have committed transactions interleaved with this user’s, those changes will be filtered out of this user’s view by splitting at that point into more than one textual comparison. Each change is thus shown once, in the Version of the user who made it, and can thus be documented once, without the confusion of interleaved changes made by others being highlighted too.

5 Version Control System integration

We add integration with external Version Control Systems using MetaEdit+’s API command-line operations and MERL (the MetaEdit+ Reporting Language for writing generators and scripts). The VCS operations are invoked from the Changes & Versions Tool, and use generators to call the necessary commands. Full implementations with generators for Git and TortoiseSVN integration are included for Windows, and users can build on these to add support for other VCSs or platforms. For instance, the TortoiseSVN implementation for versioning consists of one line for “svn update” and another for TortoiseSVN’s commit command, plus some batch file housekeeping.

5.1 What to version and how

A user can make changes and commit in one or more transactions, and when ready to version, can press ‘Save Version’ in the Changes & Versions Tool. The VCS integration puts the current state into the VCS working directory, and makes a VCS version from there.

The VCS working directory is kept separate from the MetaEdit+ directory, because VCSs cannot work sensibly with a live set of database files. MetaEdit+ copies the repository into a `versionedDB` subdirectory of the VCS working directory. (The repository’s binary files’ history compresses twice as well with standard Git or SVN deltas than if 7-Zipped first.) MetaEdit+ also writes the current textual snapshots of each graph, and in a `metamodel` subdirectory textual snapshots of the metamodels and generators. These textual files add about 10% to the initial size, but as only small parts change in subsequent versions, the ongoing increase is significantly less. The improvement in the ability to compare versions within the VCS itself is well worth it.

The standard way of using command-line VCSs – manually change a file, and in a separate command manually tell the VCS that the file has changed – is not necessary in a situation where a tool can guarantee that what is in the working directory is the exact state of the next version. Only a few VCS commands are thus needed, and MetaEdit+ will handle calling them. The commands are specified in a few MERL generators called by the various phases of the Save Version action. The bulk of the generators are for any VCS, with any variations isolated into generators with a suffix for the VCS name – e.g. `_vcsCheckIn_git()` checks in the current working directory contents, adding and committing them locally, and syncing and pushing them to a remote Git repository shared by all users of this MetaEdit+ repository.

Having the commands in MERL generators allows users to add integration for new VCSs, and to tweak existing integration if necessary. For instance, the default integration follows most customers' wishes in not including source code generated from the models, but if a particular customer wanted that, adding it for all VCSs would be a single line in the generic `_vcsCheckIn()` generator. Another customer might choose to generate source code into a separate VCS repository.

We want this new version to succeed the latest version in the VCS (regardless of who made that), and not the previous version made by this user. We thus perform an update or reset operation on the local VCS working directory before versioning, to bring the local VCS's view up to date. Since the MetaEdit+ multi-user server has already integrated the work of multiple users, the state of the repository seen by this user is exactly what we want in the next version, and we can simply write it to the working directory as detailed above. Before writing we empty the working directory, allowing us to avoid old files being left when graphs etc. have been deleted. There are thus no merges or conflicts, and we can simply commit the state of the working directory as the next version in the VCS. Repository, local and remote VCS stay in sync.

In this way, versioning in a multi-user environment is as easy as with a single user (see video[5]). The multi-user repository already makes sure we have all the other users' changes, and there is no need for the user to manually fetch others' changes, deal with diff, merge and conflicts, or think about any of these details when versioning: one click is enough to publish as the next version in a single, consistent trunk. This compares favourably with the 7–11 operations and choices the user needs to make correctly in the best Eclipse implementations (Section 3.1 above).

6 Conclusions

Over the last 15 years, much effort has been expended on improving collaboration and versioning with models. Two starting points were on offer: "clone and merge" with file-based models under a VCS, and "share and lock" with multi-user modelling tools. The vast majority of research effort has concentrated on the "clone and merge"

5 https://youtu.be/nvGQlt8dqjI

branch, and the results obtained can be considered to approach the local maximum of functionality and local minimum of complexity on that branch. This paper presents work on the less-investigated "share and lock" branch, starting with a multi-user modelling tool and adding comprehensive model comparison functionality and integration with any VCS. The results seem promising in offering similar functionality to the best examples on the "clone and merge" branch, but at a significantly lower level of complexity to the user. The number of explicit manual user operations needed to version is an order of magnitude lower than with EMF Compare and EGit, helping modellers to stay focused on their primary task: modelling.

In today's world of near-ubiquitous internet access, the requirement of a network connection to a multi-user repository is not a large one. For the rare cases where that is not possible, future work will look at adding the 3- or 4-way merges from tools like EMF Compare or CDO[6], allowing offline work to be more easily integrated.

Acknowledgments. We would like to thank MetaCase's Risto Pohjonen and Juha-Pekka Tolvanen for implementation work and discussion, and Jordi Cabot for making an early version of this material available for comment on his blog.[7]

References

1. Kolovos, D., Rose, L., Matragkas, N., Paige, R., Guerra, E., Cuadrado, J., De Lara, J., Ráth, I., Varró, D., Tisi, M., others: A research roadmap towards achieving scalability in model driven engineering. In: Proceedings of the Workshop on Scalability in Model Driven Engineering, p. 2 (2013)
2. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. International Journal of Web Information Systems 5, 271–304 (Aug 2009)
3. Brosch, P., Kappel, G., Langer, P., Seidl, M., Wieland, K., Wimmer, M.: An Introduction to Model Versioning. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18–23, 2012. Advanced Lectures. Springer Berlin Heidelberg, Berlin, pp. 336–398 (2012)
4. Cicchetti, A., Ciccozzi, F., Carlson, J.: Software Evolution Management: Industrial Practices. In: Proceedings of the 10th Workshop on Models and Evolution co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, October 2, 2016., pp. 8–13 (2016)
5. Burden, H., Heldal, R., Whittle, J.: Comparing and contrasting model-driven engineering at three large companies. In: Proceedings of the 8th ACM/IEEE

[6] https://eclipse.org/cdo/
[7] http://modeling-languages.com/smart-model-versioning/

International Symposium on Empirical Software Engineering and Measurement, p. 14 (2014)

6. Ohst, D., Kelter, U.: A Fine-Grained Version and Configuration Model in Analysis and Design. In: Proceedings of the 18th International Conference on Software Maintenance, pp. 521–527 (2002)
7. Gómez, A., Benelallam, A., Tisi, M.: Decentralized Model Persistence for Distributed Computing. In: Proceedings of the 3rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 23, 2015., pp. 42–51 (2015)
8. Murta, L., Corrêa, C., Prudêncio, J., Werner, C.: Towards Odyssey-VCS 2: Improvements over a UML-based Version Control System. In: Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models, New York, NY, USA, pp. 25–30 (2008)
9. Shahrokni, A., Söderberg, J.: Beyond Information Silos Challenges in Integrating Industrial Model-based Data. In: Proceedings of the 3rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 23, 2015., pp. 63–72 (2015)
10. Brun, C., Pierantonio, A.: Model differences in the Eclipse modeling framework. UPGRADE, European Journal for the Informatics Professional 9, 29–34 (2008)
11. Kalliamvakou, E., Palyart, M., Murphy, G., Damian, D.: A field study of modellers at work. In: Proceedings of the Seventh International Workshop on Modeling in Software Engineering, pp. 25–29 (2015)
12. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: Proceedings of the 8th International Conference on Advanced Information System Engineering, pp. 1–21 (1996)
13. Kelly, S.: CASE tool support for co-operative work in information system design. In: Information Systems in the WWW Environment, IFIP TC8/WG8.1 Working Conference, 15–17 July 1998, Beijing, China, pp. 49–69 (1998)
14. Boersma, M.: Language Workbench Challenge 2014. Available at: http://www.languageworkbenches.net/2014/08/lwc2014-the-participants/
15. Daniel, G., Sunyé, G., Benelallam, A., Tisi, M.: Improving Memory Efficiency for Processing Large-Scale Models. In: Proceedings of the 2nd Workshop on Scalability in Model Driven Engineering co-located with the Software Technologies: Applications and Foundations Conference, BigMDE@STAF2014, York, UK, July 24, 2014., pp. 31–39 (2014)
16. Bendix, L., Emanuelsson, P.: Collaborative work with software models-industrial experience and requirements. In: International Conference on Model-Based Systems Engineering, 2009. MBSE'09, pp. 60–68 (2009)