Experiences in using Optimisitic Locking in Fujaba

Christian Schneider
SE, Universität Kassel
Wilhelmshöher Allee 73, 34121 Kassel
christian.schneider@uni-kassel.de

Albert Zündorf
SE, Universität Kassel
Wilhelmshöher Allee 73, 34121 Kassel
albert.zuendorf@uni-kassel.de

1. REQUIREMENTS ON VERSIONING

In software engineering, we have profound experiences in using versioning systems for textual programming languages. For small teams one might use versioning systems with pessimistic locking like SCCS or RCS. This means, changing files is mutally exclusive. On larger teams, such pessimistic locking approach runs into problems, when more frequently multiple users need write locks on multiple files. To overcome these limitations, versioning systems with optimistic locking, such as CVS [4] or SVN [5], have been introduced. These systems allow for concurrent changes on multiple local copies of the same file. The assumption is, that different users will most likely modify disjunct parts of the file and that merging these changes will work smoothly. Before a change can be committed to the central versioning repository, the user has to merge changes already committed by other team members into his local copy. If the diffent users have modified distinct parts of the file, only, these changes will be combined in the optimistic assumptions that they do not interfere.

1.1 Vital features

Although, at the beginning most users are concerned about the automatic merging, experiences show that optimistic locking and merging works surprisingly smoothly. Using optimistic locking enables us to sandbox the team members and their different tasks very effectively. Due to our experiences, versioning with optimistic locking is indispensable for the development of large UML models, too. Large UML models will most likely be developed by teams of modelers, too. Most likely it is not possible to structure the UML model such that each team member works on a distinct part of the model without the need to modify another's part and without beeing interfered by modifications on other parts. As far as our experience goes, almost any use case requires changes on multiple classes and methods in multiple packages or components crosscutting multiple architectural layers. Thus, working on a shared model e.g. managed in a common database where each change becomes visible to all team members, instantly, results in lots of disruptions of one team member's work by the work of the others. It will be hard to find a moment in time where the model is consistent e.g. in order to generate code from it.

Thus, we strictly vote for versioning support for UML models that provides optimistic locking. Within our UML case tool Fujaba we have developed such an optimistic versioning support, using the CoObRA library (*Co*ncurrent *Ob*ject *R*eplication fr*A*mework, [2, 1, 3]). We use CoObRA for several years now within the development of the Fujaba Tool Suite itself and within student projects.

1.2 Versioning framework

Fujaba provides a sophisticated versioning support via CoObRA. Basically, CoObRA subscribes itself as listener to all objects of a given model. Then, CoObRA protocols all model changes together with undo information. Additional CoObRA operations allow to group incremental changes into user actions. Based on these information, CoObRA is already able to provide undo/redo, persistency and versioning functionality. The CoObRA system is also able to send chunks of change protocols to a project repository. This allows for team support. One user may check in his changes and other users may download these changes into their local version and just replay the change protocol. In order to identify the states of the different local models, CoObRA adds version marks to the change protocols. These version marks allow to identify the changes which have already been downloaded by some user and which have not. Note, CoObRAs versioning is based on object identifiers that are unique over sites and sessions. To achieve this, the server provides unique session ids which are used as prefixes for object ids.

As mentioned, CoObRA provides an optimistic locking concept. On update, the local model is rolled back to the state of the last synchronization. Then, the changes received from the server are applied. This creates no conflicts since the local changes have been rolled back. Next, the local changes are replayed. This might create merge conflicts, e.g. if a locally modified object has been deleted on the server site or if a locally modified attribute has been modified on the server, too. In case of a conflict, all changes of the corresponding user interaction are rolled back and the merge mechanism continues with the next local user interaction. This might result in additional conflicts, e.g. if subsequent changes rely on objects that would have been created by a rolled back user interaction. CoObRA collects all conflicting changes and provides a list of these for the application. The application may then display a user interface which shows the conflicting changes in a user friendly visualization, and which enables the user to resolve the conflict, manually.

The Fujaba Tool Suite uses the CoObRA versioning mechanism in two ways. First, Fujabas code generation provide dedicated support in order to add CoObRAs versioning sup-

port to applications modeled with Fujaba. Second, Fujaba uses CoObRAs versioning mechanism for its internal meta model, too. In the latter case, we use a very simple user interface for check in, update, and for the visualization of conflicting changes.

2. EXPERIENCES

We use CoObRAs versioning for Fujabas internal model since several years now. We have two main projects with serveral 10,000 lines of code generated from the models, the CodeGen2 and the OBA project. The first specifies and realizes and bootstraps Fujabas code generation. The second is an industrial project developing an expert system for the design of wiring harnesses of modern cars. In both projects, the number of developers is relatively small and thus, we observed only very few merge conflicts. In addition, the developers within these projects are very familar with Fujabas metamodel and thus, it was easy for them to interpret the simple conflict reports provided by CoObRAs standard mechanism.

In addition, we have used CoObRA to version the Fujaba projects of our students in several UML and modeling courses at University of Kassel. This course is run in the 4th or 5th term and it is mandatory for all computer science students. We have about 80 students per course and we organize them in small teams of about 5 students. Each team has to develop a UML model of some board game with a reasonable complexity. The model has to cover class diagrams and behavior specifications for method bodies and for unit tests. From these specifications, all code is generated, automatically, and the unit tests are run to validate the model behavior. Within these student projects, the team members most likely interfere within the common class diagram. Accordingly, we observe a certain number of merge conflicts due to renaming and deletion of common items. Another kind of conflict occurs, if two team members introduce a new class or attribute or association for the same purpose, independently. For CoObRA, this is no conflict and the merge works smoothly. Actually, the team members have to unify these model elements, manually, in order to achieve a collaboration of their individual contributions.

2.1 Merge Conflicts

In principle, merge conflicts are a major threat to your project. In practice, we deal quite well with them. One problem is the presentation of merge conflicts to the user and how the user might be supported in resolving them. Currently, Fujaba provides only very cryptic reports on refused changes. Astonishingly, even our students are able to deal with our cryptic reports quite well. For example, they interpret the message

```
alter field de.fujaba.uml.UMLClass("Student").name
from "Person" to "Collegian" failed
```

easily, since they usually recall that they just have changed the name of the “Person” class to the new name “Collegian” and some team mate has probably renamed the same class to “Student”.

Of course, merge conflicts may result in model inconsistencies. For example, one may have added a lot of object instances of some Class C to his or her model and some team mate may have removed class C. In Fujaba this results in a merge conflict since the instanceOf link connecting the object instance and its declaring class cannot be inserted into the merged project. Therefore, all these object insertions are refused. Luckily, the deletion of class diagram elements get more seldom when the project matures.

Merging may also generate context sensitive model inconsistencies. For example, two team mates may concurrently introduce a new class with the same name e.g. Person. Since CoObRA’s versioning is based on internal ids, after merging the project will contain two classes with the same name. In general, merging may result in model states that could not be reached by usual editing. Any consistency constraint that is usually maintained by the user interface may be violated. Thus, merging sometimes corrupts the internal object structures of our tool. To deal with these kinds of problems, we steadily enhance our metamodel and tool in order to allow temporary inconsistencies. As the consistency of the model still needs to be ensured, measures to fix violated consistency constraints should be provided lateron. Currently, we provide some repair mechanisms that resolve typical model inconsistencies on request. Finally, our dynamic object browser eDOBS enables our experts to repair any internal model problems.

3. CONCLUSIONS

After all, we strongly believe that versioning of UML models is absolutely necessary. This versioning must provide optimistic locking mechanisms and merging mechanisms as it is state-of-the-art for textual documents. Our experiences with Fujaba and CoObRA show that versioning of UML- and other object oriented models with optimistic locking is feasible. Of course, there are a number of pitfalls. For using CoObRA the metamodel must provide means to protocol changes, e.g. a notification mechanism. The metamodel must not rely on too many additional consistency constraints, or must be able to deal with otherwise refused changes. Finally, merging may result in conflicts, some kinds of inconsistencies, and editing operations may get lost due to such problems and the project may reach inconsistent states that could not be reached via usual editing operations. However, due to our experiences the benefits of being able to have local working copies for the members of a large project team, working on a common UML modeling project, outweigh the drawbacks multiple times. Within Fujaba, we can just not imagine, how someone can live without versioning support and optimistic locking for his or her models.

4. REFERENCES

[1] C. Schneider. CASE Tool Unterstützung für die Delta-basierte Replikation und Versionierung komplexer Objektstrukturen (Diploma Thesis, german), 2003.

[2] C. Schneider, A. Zündorf, and J. Niere. CoObRA - a small step for development tools to collaborative environments. In *WSEE*. ICSE 2004, Scotland, 2004.

[3] CoObRA 2. http://www.se.eecs.uni-kassel.de/se/?coobra, 2006.

[4] CVS - Concurrent Versions System. http://www.nongnu.org/cvs/, 2006.

[5] SVN - Subversion. http://subversion.tigris.org/, 2006.